TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

Tutkintotyö

Niko Vesa

Symbian S60 -ohjelmointi ja TKL-aikatauluohjelman toteutus

Työn valvoja Lehtori Jari Mikkolainen, TAMK
Työn ohjaaja System Engineer Jussi Moisio, Sasken Finland Oy
Tampere 2008

Tietotekniikan koulutusohjelma, Ohjelmistotekniikka
Niko Vesa

Tutkintotyö	Symbian S60 -ohjelmointi ja TKL-aikatauluohjelman toteutus 46 sivua
Työn valvoja	Lehtori Jari Mikkolainen, TAMK
Työn ohjaaja	System Engineer, Jussi Moisio, Sasken Finland Oy
Helmikuu 2008	
Hakusanat	Symbian, S60, Carbide.c++, Mobiiliohjelmistot

Tiivistelmä

Työn tarkoituksena oli tutustua Symbian-ohjelmointiin S60-ohjelmistoalustalle käyttäen Carbide.C++-ohjelmointiympäristöä ja toteuttaa sillä käyttökelpoinen prototyyppiohjelma. Erityinen kiinnostuksen kohde oli Carbiden UI Designer eli graafinen työkalu käyttöliittymän toteuttamiseen. Painopisteitä lisäksi olivat tutustuminen Symbian C++:n ohjelmointitapoihin sekä S60-ohjelmistoalustan ominaisuuksiin.

Työn alkuosassa käydään läpi Symbian C++-ohjelmoinnin ominaispiirteitä ja osaltaan verrataan C++-standardin mukaiseen ohjelmointiin. Työn keskivaiheilla tutustutaan S60-ohjelmistoalustaan, muun muassa sen käyttöliittymään ja selvitellään, kuinka turvallisuusnäkökohdat on otettu alustassa huomioon ja kuinka se vaikuttaa ohjelmien kehittämiseen. työn loppuosassa kerrotaan TKL-aikatauluohjelman suunnittelusta, toteutuksesta sekä esitellään johtopäätöksistä.

Työssä asetetut tavoitteen TKL-sovelluksen ominaisuuksille toteutuivat tyydyttävästi. Kaikki ominaisuudet, jotka suunnitteluvaiheessa ohjelmaan haluttiin, saatiin toimimaan suunnitellulla ja halutulla tavalla. Carbide.c++:n UI designer osoittautui hyväksi työkaluksi, mikäli halutaan toteuttaa nopeasti yksinkertainen käyttöliittymän peruskäyttöliittymäkomponentteja käyttäen. Vaativampaan ohjelman tekemiseen se ei vielä riitä.
Työ antaa hyvän kuvan Symbian S60 -ohjelmoinnin perusteista siitä kiinnostuneille. Se olettaa lukijan omaavan perustiedon olio-ohjelmoinnista. työssä käytetyt Carbide.c++ versiot olivat 1.2 pro ja 1.3 pro beta.

Engineering Thesis	Symbian S60 -programming and TKL-time-table program 46 pages
Thesis supervisor	Lecturer Jari Mikkolainen, TAMK
Thesis instructor	System Engineer, Jussi Moisio, Sasken Finland Oy
February 2008	
Keywords	Symbian, S60, Carbide.c++, Mobile software

Abstract

The purpose of this Thesis is getting to know Symbian programming with S60 software platform using Carbide.c++ IDE and to program S60 software with it. There was a special interest to in getting to know Carbide.c++ UI Designer. Added to this the focus was to get know Symbian coding and features of the S60 platform.

In early part of the thesis there is an introduction to the Symbian C++ programming and a comparison of how it differs to the traditional C++ programming. Later in the middle part we get to know S60 platform's program structure, UI, security and signing processes. And finally, in the last part the thesis describes the design and implementation of the TKL-application. Thesis ends to the conclusions about the application.

The goals to the TKL program set in the design phase where met in the end and valuable knowledge was obtained. The Carbide.c++ UI Designer was found to be a fast way to create simple skeleton UI using standard S60 UI components. For more demanding and/or commercial use it cannot be recommend, at least yet.

The thesis is planned to give a good overview of developing software to Symbian S60 for beginners and those who are interested. It also presupposes basic knowledge of object-oriented programming from its viewers.

LYHENTEIDEN JA MERKKIEN SELITYKSET

FP	Feature Pack. Inkrementaalinen lisäpaketti S60 -versioihin. Sisältää vähemmän uudistuksia kuin uusi S60 -versio.
VGA	Video Graphic Array. Tarkoittaa 640*480 pikselin resoluutiota.
HTML	Hyper text markup language. Kuvauskieli, josta mm. websivut rakentuvat.
EDGE	Enhanced Data rates for GSM Evolution. Tarkoittaa 2.5 sukupolven tiedonsiirtoteknologiaa mobiililaitteissa.
WCDMA	Wideband Code Division Multiple Access. Kolmannen sukupolven tiedonsiirtoteknologia mobiililaitteissa.
UIQ	User Interface Quartz. Symbian OS:ssä toimiva ohjelmistoalusta. Motorolan ja SonyEricsonin käyttämä ja omistama.
S60	Series 60. Nokian kehittämä ohjelmistoalusta Symbian OS:lle. Lisensoijia ovat mm. Samsung ja LG.
TKL	Tampereen kaupungin liikennelaitos.
LSK	Left softkey. Vasen valintanäppäin.
RSK	Right softkey. Oikea valintanäppäin.
MSK	Middle softkey. Keskimmäinen valintanäppäin. Käytetään myös nimitystä Selection key tai OK-key.
DLL	Dynamic link library.
SDK	Software development kit.

Carbide.c++	Symbian-ohjelmointityökalu, joka perustuu Eclipseen.
UI Designer	Carbide.c++:n graafinen työkalu käyttöliittymien tekemiseen.
IDE	Integrated development environment. Ohjelmointiympäristö.
Kyky	S60-ohjelmistoalussa pääsyvaltuutus arkaluontoisiin järjestelmäresursseihin.

1. JOHDANTO

Symbian-käyttöjärjestelmä johtaa markkinaosuudellaan älypuhelinmarkkinoita tällä hetkellä isolla erolla Windows ME- ja Linux- pohjaisiin ratkaisuihin. Suurimmassa osassa Symbian-puhelimista on Nokian S60 -ohjelmistoalusta. Yksi vaihtoehto sille on SonyEricsonin ja Motorolan käyttämä UIQ-ohjelmistoalusta. Viime aikoina Nokia on ryhtynyt valmistamaan näitä älypuhelimia entistä enemmän myös keskihintaluokkaan. Samalla se on lanseerannut kaksi alabrändiä, N- ja E-sarjan. Näistä ensimmäistä se kutsuu multimediatietokoneiksi ja toinen on suunnattu yrityspuolelle. Tarjonnan kasvaessa Nokia vaatii ja tarjoaa myös kehittäjille uusia mahdollisuuksia ja työtä. Vaativana pidetty Symbian-käyttöjärjestelmälle kehittäminen kaipaa kokoajan uusia osaajia tyydyttämään yritysten tarpeita alalla. Myös kehitystyökalujen kehittyminen on luonnollinen osa kehitystä.

Tutkintotyön tavoitteena on suunnitella ja kehittää TKL-aikataulusovellus Nokian S60 3rd Edition -ohjelmistoalustalle. Sovellus kykenee hakemaan aikataulutiedot TKL:n nettisivuilta ja näyttämään tiedot html:stä niin kuin ne näkyisivät web-sivulla. Tiedot tallennetaan muistiin ja ovat siten käytettävissä ilman erillisiä latauksia tai yhteyttä verkkoon. Tämän yleishyödyllisen sovelluksen on tarkoitus tarjota käyttäjälle kätevä, nopea ja aina mukana kulkeva tapa tarkastella TKL:n aikatauluja.

Työssä on tarkoitus käyttää uusinta työkalua Nokian S60 -ohjelmistoalustalle, eli Nokian syksyllä 2006 julkaisemaa Carbide.c++ Professional Editionia. Carbide.c++ perustuu ilmaiseen Eclipse-ohjelmointiympäristöön. Lisäksi Carbide.c++-perheeseen kuuluu Express ja Developer Edition Carbide.c++. Näistä kolmesta vaihtoehdosta Express Edition on maksuton. Maksulliset versiot tuovat monia kehitystä ja testausta helpottavia ja nopeuttavia ominaisuuksia perheeseen, kuten graafisen käyttöliittymän suunnittelutyökalun (UI designer) sekä testausmahdollisuuden oikeilla puhelimilla. Professionalissa on lisäksi mm. suorituskyvyn mittaamiseen tarkoitettu työkalu.

Näistä uuden Carbiden.c++:n ominaisuuksista UI designeriä käytetään hyväksi sovelluksen kehittämisessä. Näin on tarkoitus nopeuttaa käyttöliittymän kehittämistä ja testata UI Designerin soveltuvuutta ohjelmien kehittämiseen.

2. OHJELMOINTI SYMBIAN-KÄYTTÖJÄRJESTELMÄLLE /1/, /2/, /3/

Tässä luvussa käsitellään Symbian-käyttöjärjestelmän (myöhemmin Symbian) ohjelmoinnin perusteita. Symbianin pääasiallinen ohjelmistokehityskieli on C++, jolla tämä käyttöjärjestelmäkin on pääasiassa kirjoitettu. Symbianille kehitettäessä käytetään monia C++- kielen ominaisuuksia, kuten perintää, kapselointia, malleja, virtuaalifunktioita ja funktioiden ylikuormittamista. Näitä ominaisuuksia ei käytetä ainoastaan sovelluksen logiikan toteuttamisessa, vaan myös rajapinnoissa. Esimerkiksi jotkut ohjelmointirajapinnat ovat abstrakteja. Näist' rajapinnoista kehittäjä voi periyttää omia luokkia ja laajentaa niiden toiminnallisuutta tarpeen mukaan. Joistakin rajapinnoista voi tehdä olioita ja käyttää suoraan tai ne voivat olla staattisen luokan funktiokutsuja, joita voidaan kutsua suoraan. Viimeisestä hyvänä esimerkkinä on *User*-luokka, jonka kutsuista suurin osa liittyy kulloiseenkin säikeeseen tai sen kekoon. Esimerkiksi *User::Alloc()* varaa muistia nimetyn säikeen keosta.

Symbianille C++-kielellä ohjelmoitaessa on kuitenkin opeteltava monia standardista poikkeavia tapoja, esimerkiksi täysin oma toteutus poikkeuksienhallinnalle. STL (*Standard Template Library*) ei ole myöskään käytössä Symbianissa, vaan siinä on joukko omia mallitettuja kokoelmaluokkia. Yksi syy standardista poikkeaviin tapoihin on saada toteutus tehokkaammaksi niukoilla resursseilla toimiville laitteille. Oma poikkeuksienhallinta johtuu C++:n standardoinnin keskeneräisyydestä, kun ensimmäistä Symbiania lähdettiin kehittämään. Jatkossa tarkastelemme tarkemmin näitä Symbianin keskeisiä tapoja kirjoittaa C++-kielellä.

2.1 Nimeämiskäytännöt

Symbian käyttää monia nimeämistapoja. Nämä antavat kehittäjälle tietoa luokista, funktioista ja muuttujista, kuinka ne käyttäytyvät ja kuinka niitä pitäisi käyttää. Lisäksi ne auttavat automatisoituja testaustyökaluja koodin tarkastamisessa.

T-Luokat ovat yksinkertaisia luokkia, joissa on tyyppinsä mukaista tietoa. Näillä luokilla ei ole dynaamisesti varattua muistia eikä ne siten tarvitse tuhoajaa. Niillä voi kuitenkin olla jäsenfunktioita, joten ne eivät ole pelkkiä tietorakenteita. Ne eivät ole myöskään välttämättä pieniä, mikä tulee ottaa huomioon varsinkin luotaessa niitä pinomuistiin. Pinomuistin ylivuodot ovat peruuttamattomia ja aiheuttavat ohjelman sulkemisen. Mikäli luokalla ei ole jäsenfunktiota, voidaan käyttää **S**-etuliitettä (*struct*).

C-Luokat eli ns. kekoluokat, periytetään kaikki *CBase*-luokasta, joko suoraan tai toisen C-luokan kautta. Tämä varmistaa sen, että purkaja tulee kutsuttua tuhottaessa luokan ilmentymä. Kaikki *CBase*-luokat luodaan kekoon sen ylikuormitetulla *New (ELeave)*-operaattorilla, joka alustaa uuden olion muistin nollaksi. Näillä luokilla pitäisi olla yksityinen (*private*) tai suojeltu rakentaja, jolla käytännössä estetään pinomuistiin luominen. Pinomuistiin luotaessa tätä rakentajaa ei kutsuta, joten sen jäsenmuuttujat alustuvat määrittelemättömällä datalla. Tämä tuottaa ongelmia, jos oletetaan olion alustuneen niin kuin kekoluokan Symbianissa kuuluu.

R-Luokat ovat luokkia, joita käytetään kahvoina resursseihin. Nämä resurssit voivat olla asiakas-palvelinarkkitehtuurissa palvelimen hallinnoimia. Tälläinen on esimerkiksi RFile, joka asiakasluokkana käyttää hyväkseen tiedostopalvelimen toteuttamaa ja hallinnoimaa resurssia. Lisäksi Symbian API tarjoaa monia resurssiluokkia kontroloimaan käyttöjärjestelmän ytimen tarjoamia resursseja, kuten säikeitä ja prosesseita. Nämä monesti pinomuistiin luotavat oliot avataan funktiolla kuten *Open()* tai *Connect()* ja suljetaan lopuksi esim. *Close()*-funktiolla tai vastaavalla. Kun resurssiluokat ovat pelkkiä kahvoja itse resursseihin, niiden tuhoaminen ei

tuhoa itse resurssia, johon se on yhdistetty. Mikäli yhteys unohdetaan sulkea, aiheutuu muisti- tai resurssivuotoon kyseiseltä palvelinprosessilta.

M-Luokat ovat ainoa keino Symbianissa toteuttaa moniperintää. Tähän viittaa luokan nimen M (=Mix). M-rajapintaluokat sisältävät ainoastaan puhtaita virtuaalifunktioita, eikä toteutusta siten ollenkaan. Jokaiselle niiden sisältämälle funktiolle kirjoitetaan siis itselle sopiva toteutus. Perintä rajapintaluokasta mahdollistaa inkluusiopolymorfismin.
Tämä toteutetaan rajapintaluokkaosoittimen avulla. Tällöin mikä tahansa luokka, joka perii kyseisen rajapinnan, voidaan laittaa tähän osoittimeen. Esimerkiksi funktioiden parametrin tyyppinä voidaan käyttää näitä, jolloin kyseinen funktio kelpuuttaa osoittimen minkä tahansa tyyppiseen olioon, joka on peritty kyseisestä M-luokasta.

N-etuliitte laitetaan nimiavaruuden nimen alkuun. Niiden tarkoituksena on koota funktioita, jotka eivät kuulu muualle, eikä niiden haluta ”saastuttavan” globaalia nimiavaruutta. Koska nimiavaruudet ovat jokseenkin uusia Symbianissa, näemme koodissa edelleen paljon niiden tarkoituksen ajaneita, pelkkiä staattisia funktioita sisältäviä yleisluokkia. Näitä ovat esimerkiksi *User* ja *Math*, joissa ei käytetä mitään etuliitettä.

Muuttujien nimeämisessä on kaksi käytäntöä. Funktioiden parametreissä ne saavat etuliitteen *a* niin kuin *argument* (argumentti), ja jäsenmuuttujien eteen tulee *i* niin kuin *instance* (ilmentymä). Paikallisten muuttujien eteen ei tarvitse laittaa etuliitettä, vaikka hyvä käytäntö tulisi näissäkin muistaa.

Vakiot saavat **K**-etuliitteen ja *enum*:n jäsenet **E**-etuliitteen, kun itse nimi saa **T**- tai **S**-etuliitteen.

Funktioiden toimintaa kuvaamaan käytetään niiden nimissä loppupäätteitä. Funktiot, jotka saattavat aiheuttaa poikkeuksen (*leave*) päättyvät –**L**. Toinen mahdollinen pääte on –**LC**, joka edellisen lisäksi jättää osoittimen siivouspinoon (*cleanup stack*). Tämä osoitin on muistettava ottaa pois siivouspinosta sopivassa kohdassa tai ohjelma kaatuu. Lisäksi voidaan

käyttää päätteitä –**D** tai –**LD**, jotka kertovat funktion ottavan vastuun oliosta, josta sitä kutsutaan. Kutsuttu funktio tuhoaa siis olion lopuksi.

2.2 Virheiden hallinta

On erittäin tärkeää hallita virhetilanteet hyvin ohjelmoitaessa varsinkin laitteille, joiden resurssit ovat pienet. Symbian tarjoaa tähän hyvät edellytykset, ja näiden ymmärtäminen ja käyttäminen on jokaiselle kehittäjälle ensisijaisen tärkeää. Esimerkiksi muistin loppuessa on tärkeää, ettei laite kaadu eikä käyttäjä menetä mitään tietoja. Muistivuodot aiheuttavat monesti ongelmia tällaisissa laitteissa ja on erityisen tärkeätä, ettei ohjelmassa tapahdu muistivuotoja, vaan kekomuistista varatut alueet tulee aina vapauttaa virhetilanteissa.

Symbian-ohjelmassa voi tapahtua kahdenlaisia virheitä: paniikkeja ja poikkeuksia. Näistä paniikeista ei pystytä toipumaan eikä julkaistavassa ohjelmassa saa tapahtua niitä. Paniikit ovat poikkeuksetta kehittäjän virheitä, kuten indeksointi taulukon ohi. Huomatessaan, että kaikki ei mene niin kuin pitäisi, käyttöjärjestelmä antaa paniikkiviestin ja sulkee ohjelman välittömästi. Paniikkiviestissä käyttöjärjestelmä kertoo sen kategorian sekä numeron, joka kertoo paniikin syyn. Tämä viesti ei saisi siis koskaan näkyä käyttäjälle.

Leave/Trap mekanismi

Poikkeuksia varten Symbianissa on kolme mekanismia, joilla voidaan hallita virhetilanteet. C++-kielestä tuttu *try-catch-throw* on korvattu kevyemmällä *leave/trap* mekanismilla. Näiden mekanismien idea on samanlainen, mutta trap vaan tekee se try- että catch-toiminnot. Makrossa TRAP(D) kutsutaan poikkeuksen antavaa funktiota ja annetaan muuttuja virhekoodin palauttamiselle. Tänä muuttuja sisältää poikkeustilanteessa virhekoodin, jonka avulla voidaan virhetilanne käsitellä. Kaikki Symbianin virhekoodit ovat negatiivisia kokonaislukuja. **L-**loppuliitteestä tiedämme, mitkä funktiot voivat antaa poikkeuksen. Joka paikassa näitä makroja ei kuiten-

kaan kannata käyttää, vaan on hyvä kerätä ne tiettyyn paikkaan käsiteltäviksi, sillä niiden suorittaminen on raskasta. Toisaalta voidaan käsitellä vain muutama virhetilanne ja jättää loput ylemmäntason käsiteltäväksi.

Siivouspino ja kaksivaiherakennin

Siivouspino on kriittisen tärkeä Symbianin resurssienhallinnalle. Sitä käytetään pääasiassa varmuuskopiona paikallisille osoittimille, jotka osoittavat keosta varattuun objektiin. Näiden paikallisten osoittimien hallintaan ei TRAP(D) riitä, koska niiden näkyvyysalue loppu, kun näiden makrojen sisältä poistutaan. Siivouspinossa olevalla kopiolla varmistutaan muistin varauksen poistamisesta poikkeuksen tapahtuessa. Perustapauksessa osoitin laitetaan siivouspinoon funktiolla *PushL(CBase*)* ja tuhotaan *PopAndDestroy()*:llä. Luokan omistamaa instanssimuuttujaa ei saa koskaan laittaa siivouspinoon. Tällöin voi tapahtua kaksinkertainen tuhoaminen. Toiseksi siivouspinoa käytetään *CBase*:sta periytymättömien luokkien, kuten esimerkiksi **R**-luokkien hallintaan. Nämä pinotaa pinoon funktiolla *CleanupClosePushL<class T>(T& obj)*. Tällöin avattu kahva resurssiin tulee suljettua siivouksen yhteydessä.

Symbianissa on myös täysin oma tapa luoda olioita kekomuistiin. Nämä yleensä *CBase*-luokasta periytyvät luokat rakennetaan kaksivaiherakentimella. Mikäli nämä luotaisiin tavallisella C++-menetelmällä, rakentajafunktiossa tapahtuvan poikkeuksen takia missään ei olisi tallessa osoitinta rakennettavaan olioon. Tästä seuraisi todennäköisesti muistivuoto. Siispä ensin pitää rakentaa dynaamista muistin varausta tarvitsematon osa ja laitetaan osoitin siivouspinoon. Tämän jälkeen kutsutaan toista vaihetta, jossa siis suoritetaan kaikki dynaamiset muistinvaraukset. Mikäli tässä tapahtuisi jokin virhe, osoitin olisi tallessa siivouspinossa ja jo varatut jäsenmuuttujat saataisiin siivottua. Toisen vaiheen jälkeen voidaan osoitin poistaa siivouspinosta.

Seuraavassa listauksessa on esimerkkikoodi kaksivaiherakentamisesta:

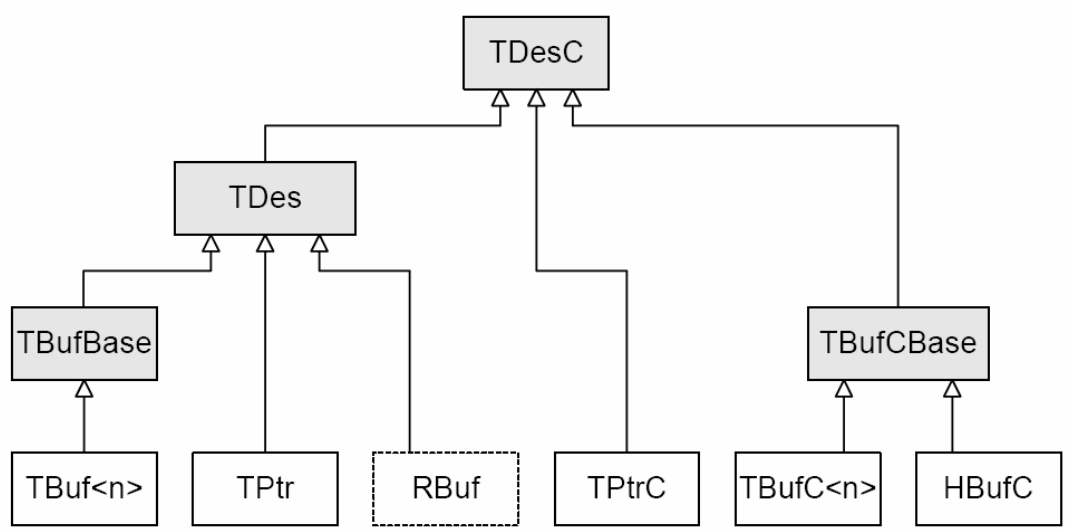
```
CWebEngine* CWebEngine::NewL(MWebClientObserver& aObserver)
{
		CWebEngine* self=CWebEngine::NewLC(aObserver);
		CleanupStack::Pop(); // self;
		return self;
}

CWebEngine* CWebEngine::NewLC(MWebClientObserver& aObserver)
{
		CWebEngine* self = new (ELeave) CWebEngine(aObserver);
		CleanupStack::PushL(self);
		self->ConstructL();
		return self;
}
```

Listaus 1. Esimerkki kaksivaiherakentamisesta.

CWebEngine-oliota luotaessa kutsutaan joko NewL()- tai NewLC()-funktiota. Kahdesta vaihtoehdosta NewL()-funktio ei jätä luotavan olion osoitinta siivouspinoon, vaan poistaa sen sieltä. Tavallisen C++-rakentajan kutsu tapahtuu NewLC()-funktiossa, jota kutsutaan NewL()-funktiosta tai voidaan kutsua suoraan, jos halutaan jättää olion osoitin siivouspinoon. Tavallisessa C++-rakentajassa voidaan alustaa kaikki staattiset luokkamuuttujat. Tämän jälkeen olion osoitin kopioidaan siivouspinoon ja kutsutaan rakentamisen toista vaihetta eli ConstructL()-funktiota. Siinä vaiheessa tehdään siis kaikki dynaamisen muistin varaukset.

2.3 Kuvaimet

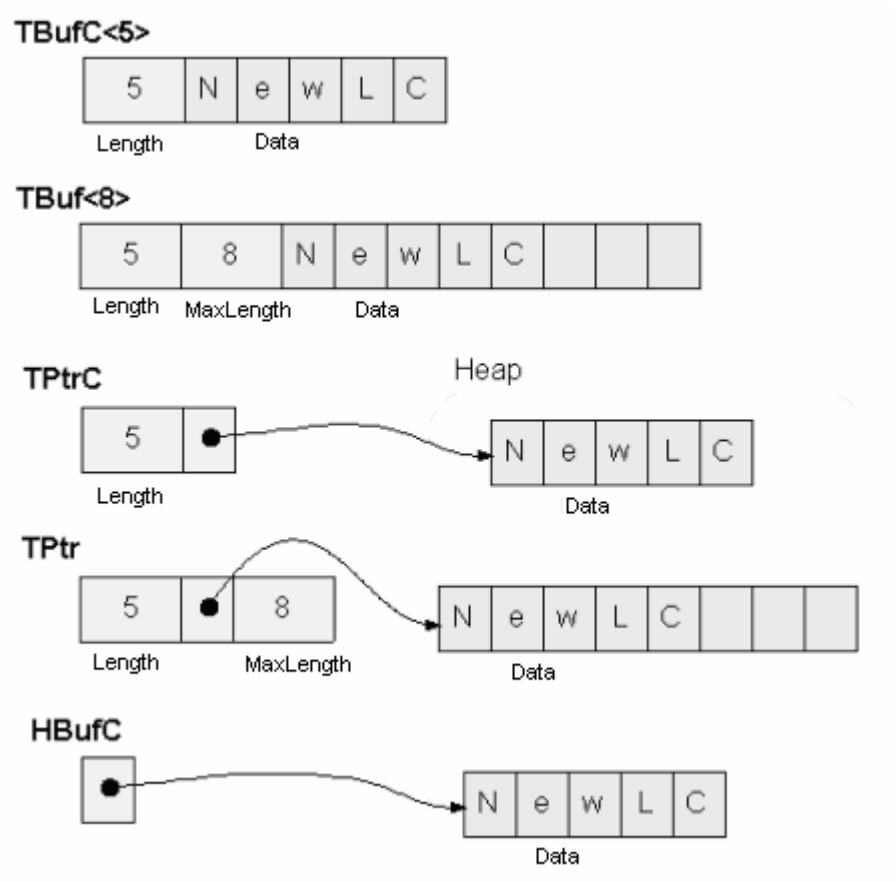
Symbianissa ei käytetä perinteisiä merkkijonoja (*string*) vaan omia kuvainluokkia (*descriptor)*. Niitä käytetään sekä teksti- että binääridatan säilömiseen, sillä ne eivät pääty binäärinollaan, kuten perinteinen C++-merkkijono. Näitä C++-merkkijonoja käytettäessä, ei voitaisi mitenkään erottaa tätä merkkijonon loppumista merkitsevää binäärinollaa binääridatasta. Kuvainarkkitehtuuri on suunniteltu vähän resursseja vieväksi, mutta vaatii tavallisiin C++-merkkijonoihin tottuneelta opettelua. Kuvassa 1 esitellään kuvaimien luokkarakenteen.



Kuva 1. Kuvaimien luokkarakenne. \5\

Kuvan viidestä alimmaisesta luokasta voidaan luoda olioita. Nämä voidaan jakaa muokattaviin tai ei-muokattaviin. Nimen perässä oleva C tarkoittaa siis sitä, että kuvainta ei voida muokata, ainakaan sellaisenaan. Kuvaimet voidaan jakaa toisellakin tapaa, nimittäin kekoon tai pinoon luotaviin sekä osoitinkuvaimiin. Nimen alussa T tarkoittaa pinoa ja H kekoa. Ptr nimessä tarkoittaa osoitinkuvainta. Osoitin tietoon sijoitetaan pinoon ja sen osoittama tieto sijaitsee keossa. TPtr:ää käytetäänkin HBufC:n muokkaamiseen. Kuvasta 2 puuttuu suhteellisen uusi kuvain RBuf, joka varaa muistinsa keosta ja on lisäksi suoraan muokattava. RBuf ei saanut nimemekseen Hbuf, koska sitä ei suoraan rakenneta kekoon, vaan se omistaa keossa sijaitsevan datan ja on vastuussa sen vapauttamisesta kun RBuf suljetaan. Sitä voisikin kuvata hyvin TPtr:n ja HBufC:n yhdistelmänä. RBuf:n helppokäyttöisyyden ja koodia selventävän vaikutuksen vuoksi sitä suositellaan käytettäväksi HBufC:n sijasta.

TBufC:llä ja TPtrC:llä on C/C++-vastineet (TBufC = char[] ja TPtrC = char*). Kuvassa 2 näkyy kuvaimien rakenne.



Kuva 2. Kuvaimien rakenne. \4\

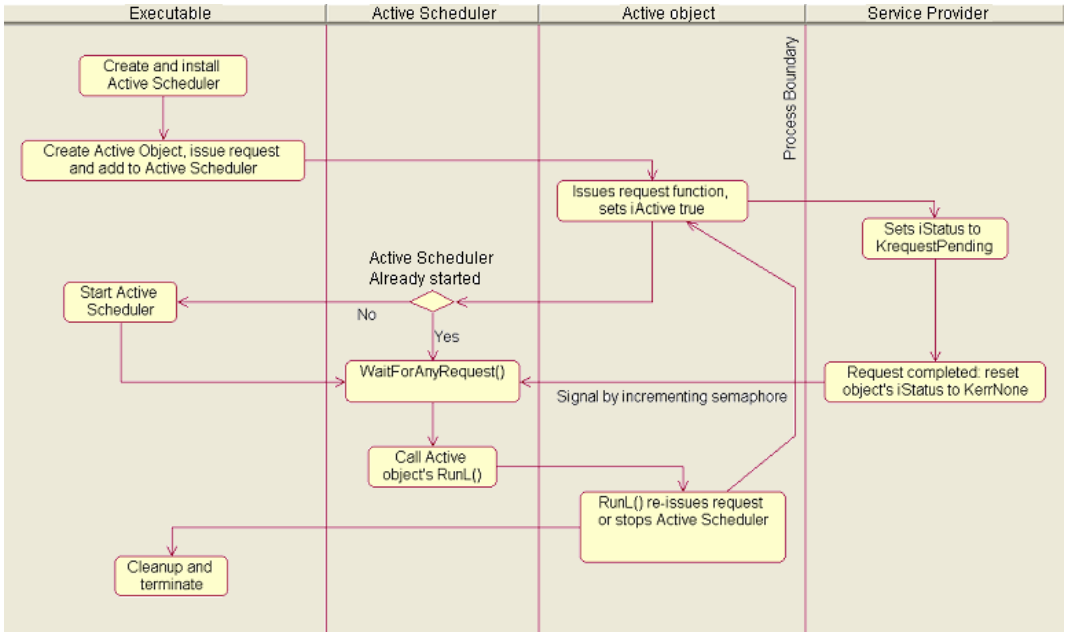
Kuvaimet tiivistetysti:

- Luotaessa TbufC-tyyppinen kuvain, annetaan sille suluissa sen maksimipituus. Yleensä annetaan siihen laitettavan tiedon pituus, koska se on ei-muokattava kuvain. Kuvaimen pituus siis tallennetaan muuttujaan tiedon lisäksi.
- Luotaessa TBuf-tyyppinen kuvain, annetaan sille sen maksimipituus ja kuvaimelle varataan sen verran muistia. Kuvaimeen tallentuu tiedon lisäksi tämä maksimipituus ja sen sisältämän tiedon pituus, joka voi siis vaihdella.
- TPtrC:hen kuuluu osoitin sen tietoon ja tämän tiedon pituus.
- TPtr sisältää myös osoittimen sen tietoon ja tämän tiedon pituuden. Lisäksi se sisältää tiedon maksimipituuden, joka voi olla isompi kuin tiedon nykyinen suuruus.
- HBufC sisältää pelkän osoittimen tietoon.

2.4 Aktiivioliot

Aktiivioliioiden avulla Symbian-käyttöjärjestelmässä voidaan suorittaa monia asynkronisia palveluita samaan aikaan yhdessä säikeessä. Käytännössähän ohjelmaa ajetaan Symbianissa yhdessä säikeessä, vaikkakin on mahdollista ajaa sitä monessa säikeessä. Aktiivioliioilla on kuitenkin mahdollista toteuttaa kaikki tarpeellinen, eikä niiden kanssa tarvitse huolehtia esimerkiksi yhteisien resurssien käytön synkronoinnista eikä semaforeista.

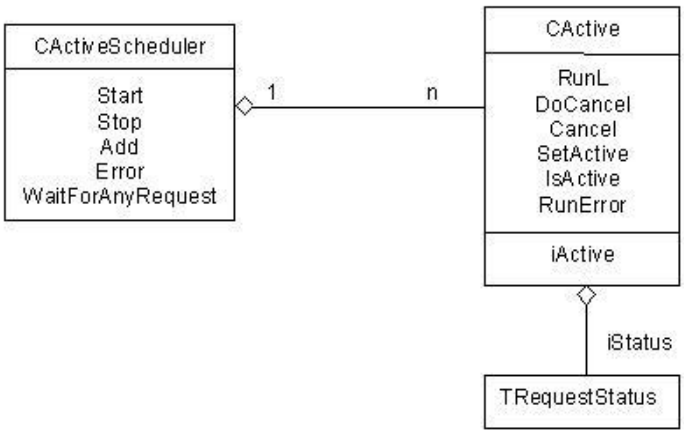
Monet Symbianin tarjoamista palveluista ovat asynkronisia, ja aktiivioliioilla niiden käyttäminen on yksinkertaista. Käyttöliittymäohjelmissa käyttöjärjestelmän sovelluskehys hoitaa aktiivivuorontimen luomisen, käynnistämisen, pysäyttämisen (Kuva 6). Asynkroniset palvelut suoritetaan taustalla toisessa säikeessä ja sen aikaa palvelupyynnön tehneen ohjelman suprittaminen jatkuu normaalisti.



Kuva 6. Aktiivivuorontimen elinkaari. /1/

Aktiivioliot täytyy lisätä aktiivivuorontimeen kutsumalla CActiveScheduler::Add() ennen palvelupyynnön tekoa.

Tämän jälkeen se odottaa palvelupyynnön valmistumista palveluntarjoajalta WaitForAnyrequest() (kuva 6). Kuvassa 7 näkyy CActiveScheduler ja Cactive luokkien funktiot kokonaisuudessa.



Kuva 7. Aktiiviolion funktiot. \6\

Asynkronisen palvelupyynnön tekevän luokan täytyy periytyä CActive-luokasta. Sen *RunL()*- ja *DoCancel*-funktiot ovat puhtaasti virtuaalisia, joten käyttäjän tulee toteuttaa ne omassa luokassaan.

- *RunL()*: *Activescheluder* kutsuu palvelun valmistuttua. Käyttäjän päätettävissä mitä halutaan sen valmistuttua tehdä. Monesti tehdään uusi asynkroninen palvelukutsu.
- *Cancel()*:Asettaa iActive-jäsenmuuttujan epätodeksi mikäli asynkroninen palvelu on käynnissä ja peruu palvelun. Mikäli palvelu ei ole käynnissä, palaa heti. Kutsuu myös *DoCancel*-funktioa.
- *DoCancel()*: Ei tulisi milloinkaan kutsua itse. Sisältää käyttäjän tekemät toimenpiteet, mikäli kesken oleva palvelu keskeytetään.
- *SetActive()*: Asettaa iActive-jäsenmuuttujan todeksi. Funktioa täytyy kutsua asynkronisen palvelukutsun jälkeen. Kutsu täytyy tehdä, jotta *Activescheluder* huomio palvelun valmistumisen.
- *RunError()*:*Activescheluder* kutsuu tätä funktioa mikäli *RunL()*-funktio palauttaa virhekoodin. Mikäli funktiota ei ole toteutettu tai palauttaa myös virhekoodin, virhe käsitellään *Activescheluder:*in *Error()*-funktiossa. Tämä taas aiheuttaa paniikin oletusarvoisesti.

Asynkronisen palvelun kutsussa välitetään myös muuttuja iStatus, jonka arvo kertoo kutsun sen hetkisen tilan, esimerkiksi KrequestPending. Tämä kertoo sen, että palvelua suoritetaan. Asynkronisen palvelun valmistuttua iStatus:n arvoksi asetetaan jokin virhekoodi, toivottavasti KErrNone. *Activescheluder* tietää palvelun valmistumisesta, kun jonkun aktiiviolion iActive-jäsenmuuttuja on epätosi ja muuttuja iStatus eri kuin KrequestPending (tosin palveluntarjoaja herättää sen aluksi inkrementoimalla sen semaforia).

2.5 Asiakas-palvelinarkkitehtuuri

Symbian-käyttöjärjestelmässä käytetään palvelimia hallinnoimaan resursseja, joita asiakkaat tarvitsevat. Palvelimet ovat ohjelmia, jotka palvelevat asiakasohjelmia(myöhemmin pelkkä asiakas) jollakin tapaa. Esimerkiksi tiedostopalvelin hallinnoi kaikkia tiedostoja ja tarjoaa niihin liittyvät palvelut kuten esimerkiksi tiedoston luonti ja poisto. Palvelimet periaatteessa odottelet komentoja asiakkailta, suorittavat ne sekä kertovat palvelun valmistumisesta ja tuloksesta asiakkaalle.

Asiakas-palvelinarkkitehtuuri mahdollistaa Symbian-käyttöjärjestelmän asynkroniset palvelut, jotka ovat keskeisiä suoritettaessa ohjelmia yhdessä säikeessä. Lisäksi kyseinen arkkitehtuuri lisää tehokkuutta, kun kaikki asiakkaat voidaan palvella yhdeltä palvelimelta. Se lisää myös turvallisuutta, kun asiakas ei voi käytännössä kaataa palvelinta.

Palvelimet käyttävät aktiivioliosovelluskehystä tiedottaakseen palvelun valmistumisesta sen sijaan, että kyseltäisiin kokoajan sen valmistumista. Näin prosessorin käyttö pienenee, ja virtaa kuluu vähemmän. Tämä on ensisijaisen tärkeää mobiililaitteissa.

3. S60-OHJELMISTOALUSTA

S60-ohjelmistoalusta on Nokia omistama, mutta se lisensoi sitä myös kolmansille osapuolille. Tällä hetkellä Samsungilta ja LG:ltä on tulossa S60-älypuhelinmalleja Nokian lisäksi. S60-ohjelmistoalustan alla on Symbian-käyttöjärjestelmä. S60 on alun perin suunniteltu yhdellä kädellä ohjattaville mobiililaitteille, joissa on näppäimistö. S60 on käyttänyt Symbian OS- käyttöjärjestelmän kolmesta viitetoteutuksesta vaihtoehdosta Pearl-viitetoteutusta, joka on suunniteltu muun muassa juuri tämän tyyppisille laitteille. Tässä osiossa käymme läpi S60:n käyttöliittymän ulkoasua sekä sen keskeisimpiä komponentteja ja niiden toimintaa. Ensiksi kuitenkin pieni kertaus S60:n historiasta.

3.1 Versiohistoria.

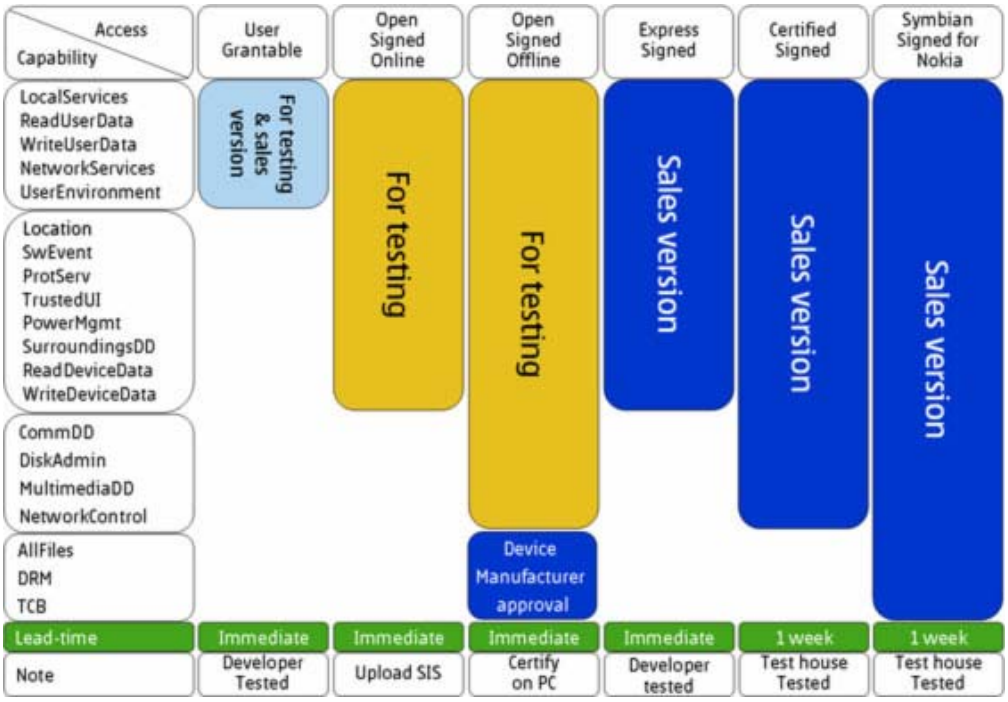
S60:n historia alkaa versiosta 1.0. Tässä käyttöjärjestelmäversio oli Symbian OS 6.1. Tätä aikaisemmat versiot kulkivat Epoc-nimellä, jonka oli kehittänyt Psion Software Ltd. Kännyköiden valmistajien ostaessa yrityksen, käyttöjärjestelmän nimi vain muuttui uusien omistajien myötä.

Ensimmäinen tällä S60 1st-editionilla markkinoille tullut puhelin oli Nokian 7650, joka oli hyvin edistyksellinen tullessaan markkinoille. Siinä oli mm. iso värinäyttö ja VGA-resoluutioinen kamera videotallennuksella. /7/
S60 2nd-editionille julkaistiin kolme lisäosaa (FP), joiden mukana tulleisiin ominaisuuksiin kuului mm. tuki html:lle, suuremmille näytön tarkkuuksille, skaalautuvalle käyttöliittymälle ja tuki edge- ja wcdma-verkoille.

Kolmas versio S60:stä rikkoi binääriyhteensopivuuden aikaisemmille versioille tehtyjen ohjelmien kanssa ja lisäsi ohjelmistoalustalle uuden turvallisuusmekanismin (*platform security*). Tästä lähtien jokainen ohjelma on täytynyt allekirjoittaa, ennen kuin se on voitu asentaa puhelimeen.

Allekirjoituksia on erilaisia ja tiettyjen rajapintojen käyttäminen vaatii ohjelmalle tietyn tasoiset valtuudet ja osa on vapaasti ohjelmien käytettävissä. Osaan rajapintoja voi luvan antaa käyttäjä asentaessaan ohjelmaan.

Lisäksi on tarjolla erilaisia allekirjoitus vaihtoehtoja, joilla ohjelma saa luvan käyttää tiettyjä toimintoja (Kuva 8). Tämä kaikki turvallisuuden nimissä, jotta erilaiset haittaohjelmat ei pääsisi valloille.



Kuva 8. Ohjelmarajapinnan kyky/allekirjoitus jaottelu. /8/

Rajapinnoista 60% on edelleen vapaasti käytettävissä. Kykyjä vaativien rajapintojen olevammassa oloa voi ajatella kahdelta kantilta. Toisaalta ne kertovat niitä käyttävän ohjelman rajoituksista sekä luvista ja toisaalta ohjelman luotettavuudesta/14/. Lähteestä 13 löytyy lyhyt kuvaus S60-ohjelmistoalustan kyvyistä. Seuraavassa kappaleessa käydään läpi tarkemmin erilaiset allekirjoitusprosessit.

3.2 Allekirjoitus prosessi

Jokainen ohjelma tarvitsee siis allekirjoittaa S60 3rd-editionista lähtien vaikka se ei käyttäisikään kyvyiden alaista ohjelmointirajapintaa. Tosin vain rajoittamatonta ja/tai *User Grantable*:n alaista koodia käyttävän ohjelman voi allekirjoittaa itse tehdyllä avaimella ja sertifikaatilla.

Muut kyvyt (kuva 8) vaativat vähintään Symbian Signed:in mukaisen allekirjoitusprosessin läpikäymisen (kuva 9). Symbian Signed tarjoaa neljä erilaista vaihtoehtoa tähän, jotka tarjoavat kehittäjille käyttötarkoitukseen sopivan vaihtoehdon.

	Publisher ID Required	Independent Testing Required	IMEI Restrictions
Open Signed Online	NO	NO	YES
Open Signed Offline	YES	NO	YES
Express Signed	YES	NO	NO
Certified Signed	YES	YES	NO

Kuva 9. *Symbian Signed* -vaihtoehdot.

Open Signed -vaihtoehdot helppokäyttöisimpiä ja soveltuvat hyvin testi-, ei-kaupalliseen- ja omaan käyttöön. Nämä allekirjoitukset ovat voimassa 36 kuukautta.

Open Signed online -vaihtoehtoa voi käyttää ilman *Publisher ID:tä*. Tämän puuttuminen rajoittaa ohjelman allekirjoittamisen kerralla vain yhdelle IMEI:lle eli ohjelmaa ei voi asentaa kuin yhteen laitteeseen. Kyseisessä prosessissa kehittäjä lähettää sähköpostilla allekirjoittamattoman SIS-tiedoston haluamansa IMEI:n ohella *Symbian Signed*:iin, josta lähetetään sähköpostilla takaisin allekirjoitettu SIS-tiedosto.

Publisher ID:n ostaneet voivat allekirjoituttaa ohjelmansa *Open Signed offline* -vaihtoehdolla enintään 1000:lle laitteelle kerralla. Tässä vaihtoehdossa ei tarvitse lähettää SIS-tiedostoa *Symbian Signediin* allekirjoituttevaksi, vaan sen voi tehdä omalla koneella. Tähän tarvitaan kuitenkin sertifikaattitiedoston lataaminen *Symbian Signedistä*. Lisäksi verrattuna online-vaihtoehtoon, näin allekirjoitetut ohjelmat saavat käyttää useampia kykyjä (kuva 8).

Publisher ID maksaa vuodessa 200$ ja sen voi ostaa niitä hallinnoivalta TC TrustCenter:ltä.

Express Signed -vaihtoehto soveltuu sekä kaupallisiin että ei-kaupallisiin ohjelmiin. Siinä ei ole IMEI-rajoituksia eikä se vaadi ulkopuolista testaamista, mutta se vaatii allekirjoitettavan ohjelman omaa testaamista ja näiden tulosten lähettämistä *Symbian Signediin*. Lisäksi tämän allekijoituksen haluava joutuu lähettämään raportin ohjelman perustoiminnallisuudesta ja perustelulla kykyjen pyytämistä. Tässä vaihtoehdossa jokainen allekirjoitettu ohjelma maksaa ja vaati *Publisher ID*:n omistamista. *Express Signed* -ohjelmia valitaan satunnaisesti auditointiin ja testiin *Symbian Signedin* puolesta ja mikäli ohjelma ei läpäise näitä, et voi käyttää *Express Signed* -vaihtoehtoa ennen kuin sinun seuraavat kaksi ohjelmaasi ovat läpäisseet *Certified Signed* -allekirjoitusprosessin. *Platform Approval* kykyjen käyttö ei ole mahdollista tällä vaihtoehdolla.

Certified Signed -vaihtoehto soveltuu parhaiten kaupallisten ohjelmien allekirjoittamiseen. Tämä allekirjoitus mahdollistaa kaikkien mahdollisten kykyjen käyttämisen ohjelmassa eikä sillä ole IMEI-rajoituksia. *Manuractu-rer Approval* -kyvyt vaativat erillisen hyväksynnän valmistajalta, jonka saaminen on erittäin vaikeata. Se vaatii myös jonkun hyväksytyn ulkopuolisen tahon testaamista ennen kuin ohjelma voidaan allekirjoituttaa. Testauttamisesta koituu myös tietenkin lisäkuluja *Publisher ID* maksun ja itse ohjelman allekirjoitusmaksun lisäksi.

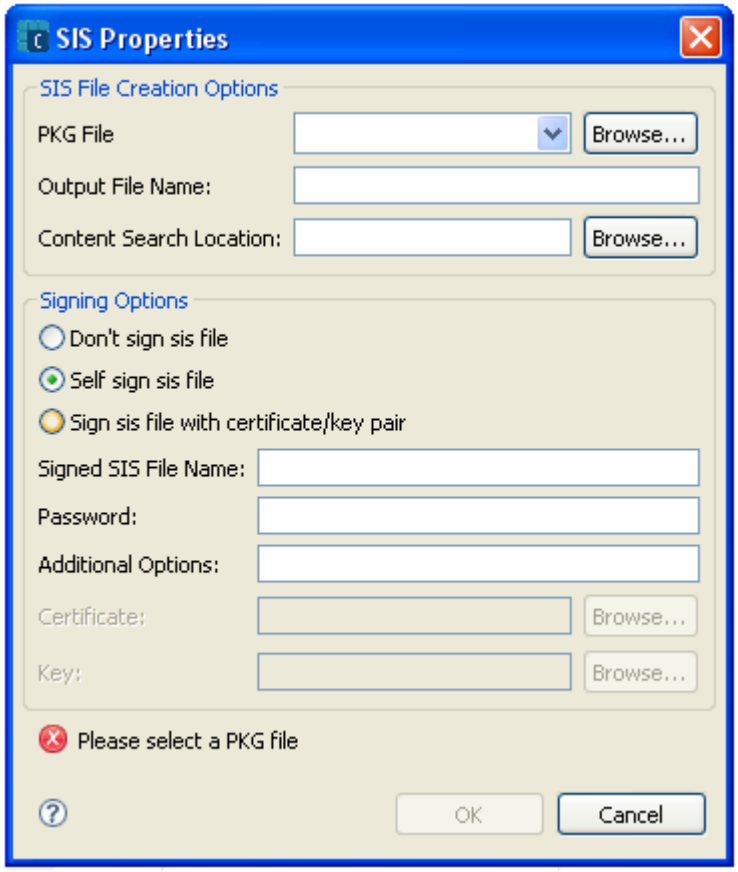
Mikäli ohjelma ei tarvitse *Symbian Signed* -allekirjoitusta, voit luoda tarvittavan avaimen ja sertifikaatin itse. Näillä voidaan sitten allekirjoittaa ohjelma. Tämän voi tehdä komentorivillä seuraavasti:

- makekeys -cert [-password <password>] [-len <key-length>] -dname <distinguised-name-string> <private-key-file> <public-key-cert>

Yllä oleva komento tuottaa kaksi tiedostoa (.key ja .cer). Näitä tarvitaan siis itse allekirjoittamiseen, jonka voi tehdä esimerkiksi komentoriviltä seuraavasti:

- SignSIS <.sis to be signed> <signed .sis> <certificate file> [key file <password>]

Komento tuottaa siis allekirjoitetun asennustiedoston, jonka voit nyt siis asentaa haluamaasi puhelimeen. Itseallekirjoitus voidaan tehdä helposti myös esimerkiksi carbide.c++ 1.3:lla, joka kutsuu automaattisesti make-keys- ja signsis-ohjelmia. Kuvassa 26 näkyy Carbiden.c++:n asettaminen käyttämään tätä ominaisuutta.



Kuva 10. Carbide.c++:n allekirjoitusasetukset

Kehittäjä valitsee ”Self sign sis file” ja muut tarvittavat tiedot (Kuva 10) ja Carbide.c++ hoitaa loput.

3.3 Käyttöliittymän ulkoasu

Tämän kappaleen kuvat käyttöliittymästä ovat S60 3rd-edionista. Käyttöliittymän näkymän sommitelma (*Layout)* koostuu kolmesta osasta eli ns. *Paneista.* Nämä ovat *Status-, Main-* ja *Control Pane*(Kuva 12). Näistä ylin eli *Status Pane* (Kuva 11) jakautuu moneen pienempään osaan. *Main Panen y*lin osa *Indicator Pane* (kuva 12) on olemassa vain käyttöliittymän valmiustilassa (*Idle state*). Ohjelmissa voidaan tämän sijasta käyttää *Battery Panea (Universal Indicator Pane)* näyttämään haluttua tietoa, kuten esim. Bluetoothin päällä olemista. /9/

Kuva 11. *Status Pane*.

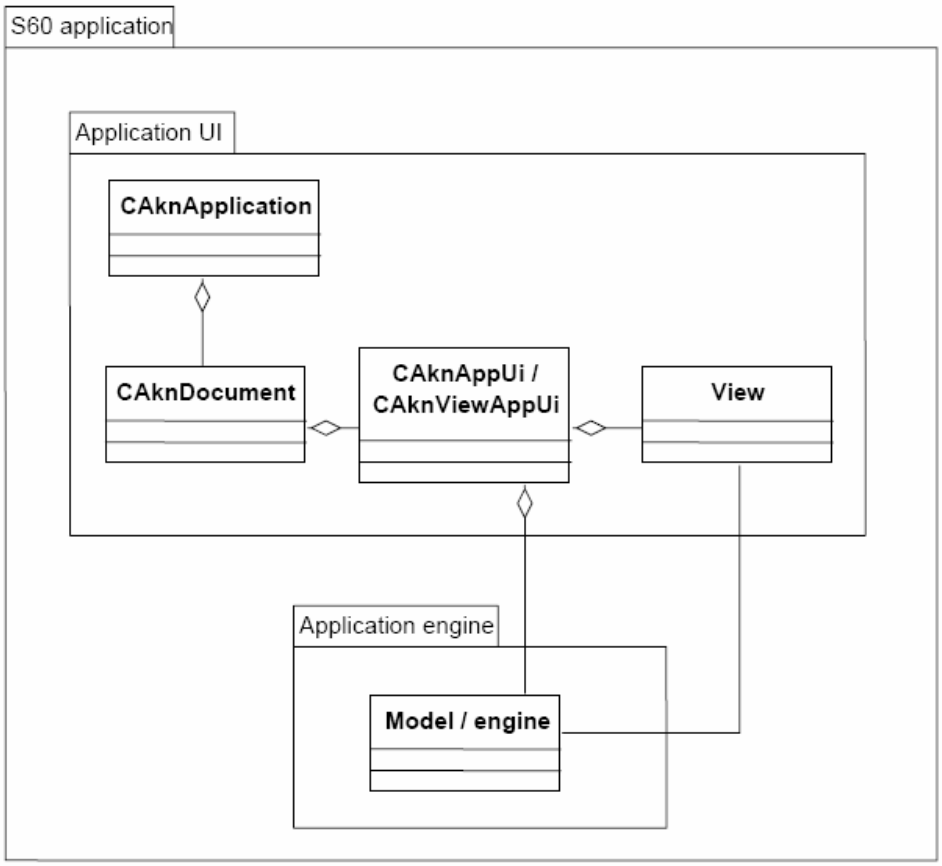
Context Panessa on tyypillisesti näkyvissä ohjelman logo ja *Title Panessa* sen nimi. *Navi Panea* voidaan käyttää moneen tarkoitukseen, kuten välilehtien näyttämiseen, halutun tekstin näyttämiseen tai ilmoittamaan ohjelman tilasta. Se voi olla myös tyhjä. *Signal Pane* näyttää verkon voimakkuuden ja lisäksi siinä voidaan näyttää kuva datayhteyden olemisesta päällä. /9/

Kuva 12. Valmiustila (*Idle State*).

Main Panessa (Kuva 12) näytetään ohjelman pääasiallinen tieto. Se voi olla esimerkiksi lista tai ruudukko muodossa. Myös kaikki dialogit, valinta-tapahtumat näytetään siinä. Siihen voidaan myös piirtää periaatteessa ihan mitä halutaan, jolloin on kuitenkin täysin ohjelman kehittäjän harteilla miltä se näyttää. *Control Panessa* (Kuva 12) on RFK, MSK (näkyvissä 3rd-edition FP 2:ssa) ja LFK. /9/

3.4 Käyttöliittymän arkkitehtuurit

Käyttöliittymäohjelman tekemiseen on kolme erilaista arkkitehtuuria joista valita kun lähdetään ohjelmaa suunnittelemaan. Näiden arkkitehtuurien toteutus vastaa pitkälti MVC-mallia, jossa ohjelma jaetaan kolmeen osaan; malliin (*Model*), näkymään (*view*) ja ohjaimeen (*Controller*). /10/ Kuvasta 13 näkyy tyypillisen S60-ohjelman rakenne ja luokkien välisen suhteet.



Kuva 13. Tyypillisen S60-ohjelman rakenne ja luokkien suhteet toisiinsa.

CAknApplication-luokasta periytetyn luokan tehtävänä on toimia ohjelman aloituskohtana ja mikäli kyseinen ohjelma on jo käynnissä, ohjata siihen avaamatta toista samaa ohjelmaa. Lisäksi se luo CAknDocument-luokasta periytetyn luokan.

CAknDocument-luokka on pohjana ohjelman tiedostojen luontiin ja pääsyyn niihin. S60:ssä ohjelman tietojen automaattinen tallennus on oletusarvoisesti pois päältä, mutta helposti toteutettavissa oleva mahdollisuus. /10/

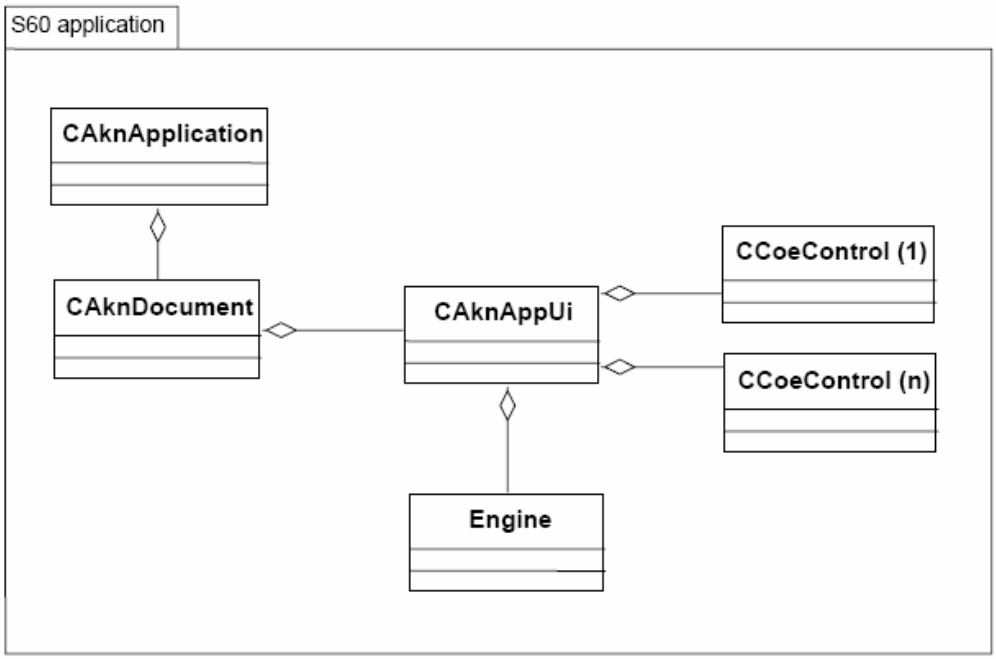
CAknAppUi/CAknViewAppUi-luokasta periytyjä luokka vastaa MVC-mallin ohjainta. Sen tehtävänä on ohjata valikko- ja näppäinkomentoja, hoitaa näkymien vaihto ja komentaa malli osaa ohjelmasta. /10/

Näkymän eli tehtävä on piirtää näkymä näytölle, vastaan ottaa käyttäjän komentoja ja ohjata ne ohjaimelle. Lisäksi se tyypillisesti seuraa mallin muutoksia ja päivittää näkymän sen mukaan. /10/

Malli edustaa ohjelman tietoja ja tilaa. Se sisältää kaiken toiminnallisuuden mikä ei liity käyttöliittymään, kuten esimerkiksi verkkoliikennöinnin tai tietokantayhteydet. Monesti malli toteutetaan omana luokkakirjastona. /10/

Perinteinen arkkitehtuuri

Perinteisessä arkkitehtuurissa tekijällä on suurin vapaus tehdä niin kuin haluaa, mutta joutuu tekemään kaiken tyhjästä ja tarvitaan paljon työtä juuri minkäänlaisen toiminnallisuuden aikaansaamiseksi. Kuvassa 14 on perinteisellä arkkitehtuurilla toteutetun ohjelman rakenteen pohjaluokkineen. /1/



Kuva 14. Perinteisen arkkitehtuurin rakenne. /10/

Ohjainluokka periytetään tässä arkkitehtuurissa CAknAppUi-luokasta, joka omistaa näkymäluokat (Kuva 14). Nämä ns. kontrolliluokat periytetään aina CCoeControlli-luokasta.

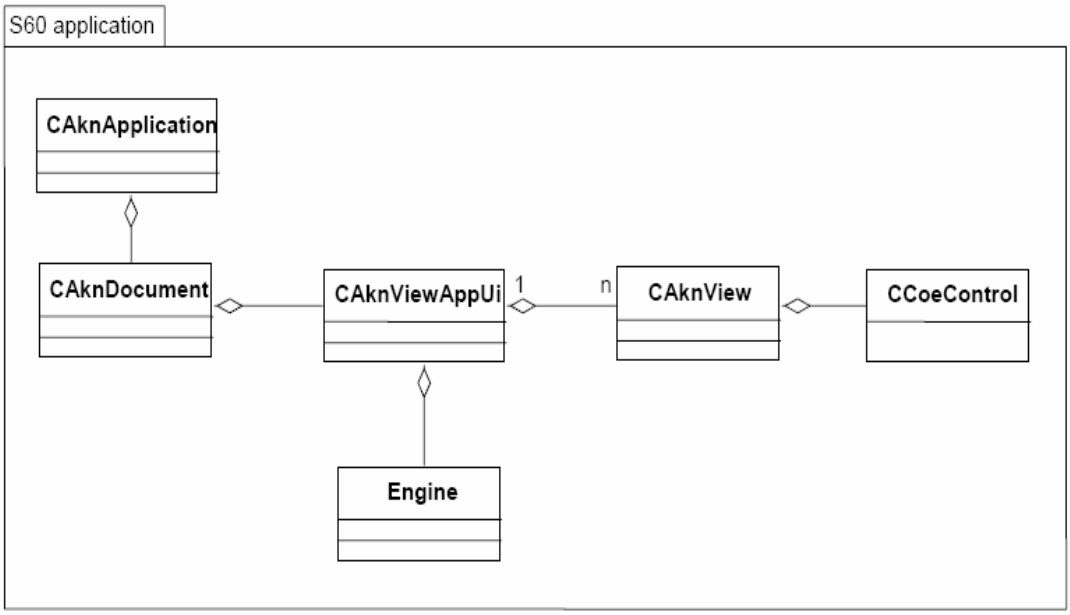
Dialogiarkkitehtuuri

Dialogiarkkitehtuurilla tarkoitetaan ohjelmaa, jossa pääpaino asioiden näyttämiseen ruudulla on dialogien vastuulla. Dialogeja käytetään paljon S60: käyttöliittymissä, mutta niiden käyttö ei välttämättä tarkoita dialogiarkkitehtuurin käyttämistä, sillä niitä voidaan käyttää aivan hyvin muiden arkkitehtuurien yhteydessä. Merkittävien heikkous dialogeilla on se, että niitä voidaan käyttää vain ainoastaan ohjelman näkymärakenteen päätepisteissä. Merkittävänä etuna voidaan mainita niiden hallinnoitavuus resurssitiedostoista, jolloin niitä voidaan muuttaa ilman että ohjelma koodia tarvitsee kääntää uudelleen. Lisäksi monisivudialogeilla on automatisoitu välilehtien toiminnallisuus ja tiedot tallennetaan automaattisesti kun välilehteä vaihdetaan. /1/, /10/

Näkymäarkkitehtuuri

Näkymäarkkitehtuurissa (Kuva 15) ohjelman ohjainluokka periytetään CAknViewAppUi-luokasta, kun perinteisässäarkkitehtuurissa se periytettiin CAknAppUi-luokasta**.** Ohjainluokka omistaa yhden tai useamman CAknView-luokasta periytetyn luokan. Näitä voidaan kutsua aliohjaimiksi. Nämä aliohjaimet ohjaavat yksittäistä näkymää. Aliohjainluokka ei itse piirrä näytölle mitään vaan omistaa CCoeControl-luokasta periytetyn luokan eli MVC-mallin näkymän. Tämän tehtävänä on piirtää näytölle. Aliohjainluokkien instanssit rekisteröidään näkymäpalvelimelle, minkä jälkeen ne voidaan helposti aktivoida varsinaisesta ohjainluokasta. Tällöin sillä hetkellä aktiivisena oleva näkymä poistetaan päältä. Aliohjainluokat saattavat helposti sekoittaa varsinaiseen MVC-mallin mukaiseen näkymään, koska niitä kutsutaan S60-maailmassa *vieweiksi*(näkymä). Näkymiä kutsutaan taas *containereiksi*. Tässä työssä käytetään MVC-mallin mukaisia nimiä eli CAknView-luokkia kutsutaan aliohjaimiksi ja container-luokkia näkymiksi. /1/, /10/, /11/

Ohjelman kaikki näkymät ovat samalla tasolla eikä niillä ole siten mitään hierarkiaa. Siirtyminen niiden välillä voidaan tietenkin toteuttaa hierarkkiseksi.



Kuva 15. Näkymän arkkitehtuurin rakenne. /10/

3.5 Kontrollipino

Ohjainluokan omistamaan kontrollipinoon lisätään ohjelman näkymäoliot, jotka peritään aina vähintään siis *CCoeControl*-luokasta. Symbianissa näitä kutsutaan kontrolleiksi, mutta kutsun niitä tässä näkymiksi, aikaisemmin esittelemäni MVC-mallin mukaisesti. Lisääminen kontrollipinoon tapahtuu kutsumalla *AddToStack*()-funktiota. Ohjelma tarjoaa kontrollipinoon rekisteröityneille näkymäluokkien instansseille näppäintapahtumia kutsumalle niiden *OfferKeyEvent*()-funktiota. Mikäli kyseinen näkymä omistaa käyttöliittymäkomponentin, välitetään se sille yleensä tässä funktiossa. Mikäli näkymä ei ole kiinnostunut tarjotusta näppäintapahtumasta, ohjelma välittää sen seuraavalle kontrollipinossa olevalle näkymälle. Näkymille voidaan antaa myös antaa prioriteetti, kun se lisätään kontrollipinoon. Tällöin korkeammalla prioriteetilla olevalle näkymälle tarjotaan näppäintapahtumaa, vaikka se olisi pinon pohjalle. Mikäli yksikään kontrolli ei kuluta näppäintapahtumaa, se tarjotaan ohjainluokalle *HandleKeyEvent*()-funktion kautta. Näkymä poistetaan pinosta kutsumalla funktiota *RemoveFromStack*(). /3/

Perinteisessä ja dialogiarkkitehtuurissa näkymien lisäys tapahtuu yleensä luotaessa ohjainluokka. Näkymäarkkitehtuurissa se tehdään CAknView-

luokan *DoActivate*()-funktiossa. Vastaavasti poisto tapahtuu AppUi-luokan purkajassa tai *DoDeActivate*()-funktiossa. /3/

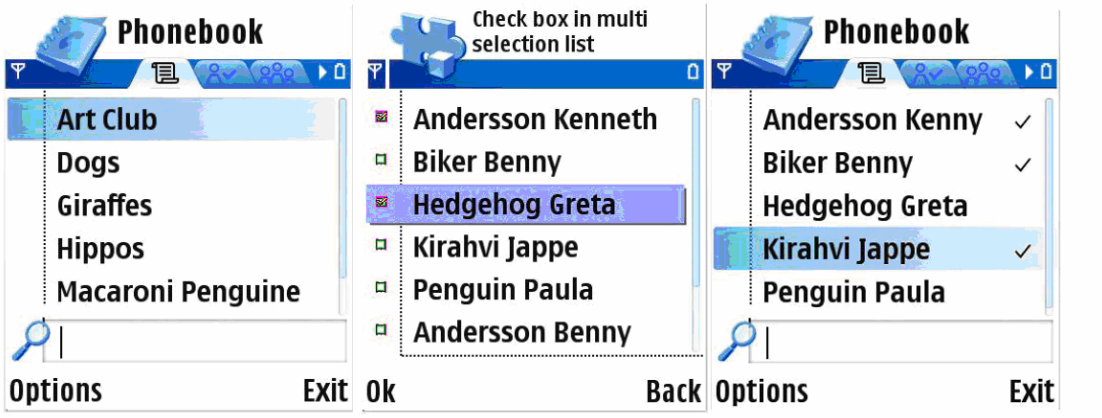
.

3.6 Käyttöliittymäkomponentit

Käyttöliittymäkomponenteilla on keskeinen osa S60-ohjelmien käyttöliittymissä. S60 tarjoaa useita valmiita komponentteja käyttöliittymän toteuttamiseen, kuten erilaisia listoja, ruudukkoja, dialogeja ja editoreja. Lisäksi ohjelmoijan voi toteuttaa räätälöityjä komponentteja. /11/

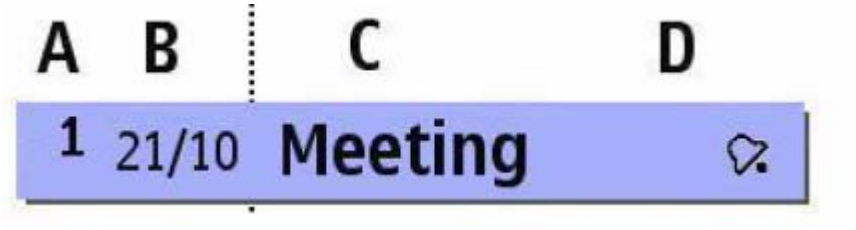
TKL-aikatauluohjelma käyttää muutamaa listakontrollia ja käsittelen tässä kappaleessa lähinnä niitä. Listakontrollit pitävät sisällään tarkoituksenmukaisen määrän alkioita näyttämään haluttuja tietoja

Valintalistakontrollista (*Selection listbox*) käyttäjä voi valita yhden alkion, kun taas monivalintalistakontrollista (*Multiselection listbox*) voidaan valita useampi. Merkintälistakontrollista (*Markable listbox*) voidaan merkitä valituksi useampi alkio. Kuvassa 16 on esimerkit näistä listatyypeistä.



Kuva 16. Valinta-, monivalinta- ja merkintälistakontrolli. /12/

Kaikki alkiot kuvassa 16 ovat tavallisia yksirivisiä alkioita. S60 tarjoaa kuitenkin monia erilaisia valmiita alkioita. Lisäksi on mahdollista tehdä omia räätälöityjä alkioita. Kuvassa 17 on listakontrollin alkion pohjapiirros.



Kuva 17. Listakontrollin alkion pohjapiirros.

Yksittäinen alkio jakautuu neljään rivistöön (kuva 17). Niiden ei ole pakko olla vierekkäin, vaan voivat sijaita myös lomittain tietyin säännöin. Näin niistä on mahdollista saada myös kaksirivisiä. Eri riveihin voidaan laittaa erilaista tietoa.

- A: Pieni kuva tai numero.
- B: Otsikko
- AB: Otsikko tai iso kuva
- C/BC/ABC: Tekstiä
- D: pieni kuva

Valikkolista on kokoelma erilaisia komentoja(kuva 18). Se aukeaa yleensä painamalla LSK:tä. Käyttäjä voi valita näistä yhden ja ohjelma suorittaa annetun komennon. Tämä valikkolista määritetään resurssitiedostoon listauksessa 2 näkyvällä tavalla.

Kuva 18. Valikkolista.

Halutun komennon käyttäjä voi valita joko painamalla valintanäppäintä tai LSK. Mahdollinen alivalikko aukeaa sellaisen omaavan komennon kohdalla, painamalla oikeaa nuolinäppäintä.

```
RESOURCE MENU_PANE r_testi_container4_menu_pane1_menu_pane
{
            items =
            {
            MENU_ITEM
            {
            command = ETestiContainer4ViewPys_kkien_lis_ysMenuItemCommand;
            txt = STR_TestiContainer4View_4;
            },
            MENU_ITEM
            {
            command = ETestiContainer4ViewValitseMenuItemCommand;
            txt = STR_TestiContainer4View_5;
            }
            };
}
```

Listaus 2. Esimerkki valikkolistan määrittelemisestä resurssitiedostoon.

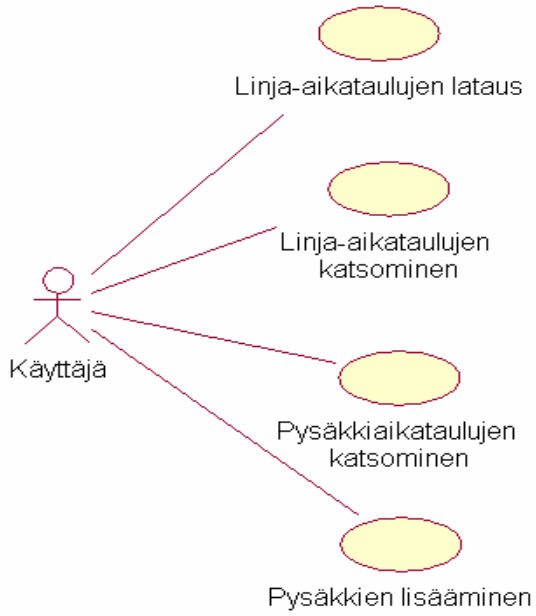
Listauksessa 2 on kaksi valikkokomentoa (MENU_ITEM), jotka pitävät sisällään itse komennon sekä valikkolistassa näkyvän tekstin lokalisaatio avaimen. Käyttäjän valittua komennon se välitetään ensiksi aliohjain luokan *HandleCommandL(TInt aCommand)*-funktion käsiteltäväksi. Mikäli komentoa ei käsitellä siinä tai käytössä ei ole näkymä arkkitehtuuri, se annetaan varsinaisen ohjainluokan samaisen funktion käsiteltäväksi. Komennolle pitää luonnollisesti toteuttaa haluamasta toiminnollisuus jompaankumpaan funktioon.

4. TKL-AIKATAULUSOVELLUS

4.1 Toiminnallisuus

TKL-aikatauluohjelman tarkoitus on antaa käyttäjälle nopea ja helppo tapa katsoa Tampereen sisäisen linja-autoliikenteen aikataulutietoja. Ohjelmalla voidaan katsoa sekä linja että pysäkkikohtaisia aikataulutietoja. Ensimmäisellä käynnistyskerralla asennuksen jälkeen, ohjelma lataa jokaisen linjan aikataulutiedot Tampereen kaupungin julkisenliikenteen palvelimelta ja tallentaa ne puhelimen muistiin myöhempää käyttöä varten. Näin tietojen näyttäminen jatkossa on nopeaa ja yksinkertaista.

Pysäkkikohtaisia aikataulutietoja käyttäjä voi ladata käyttöönsä valitsemalla haluamansa pysäkit erillisestä valikosta. Tällöin aikataulutiedot tallennetaan puhelimen muistiin, kuten myös linjojen aikataulutiedot ensimmäisen käynnistyksen yhteydessä. Ne ovat siten myös helposti käytettävissä, kun käyttäjä haluaa katsoa tietyn pysäkin aikataulutietoja. Näistä saadaan neljä käyttötapausta jotka näkyvät myös kuvasta 19.



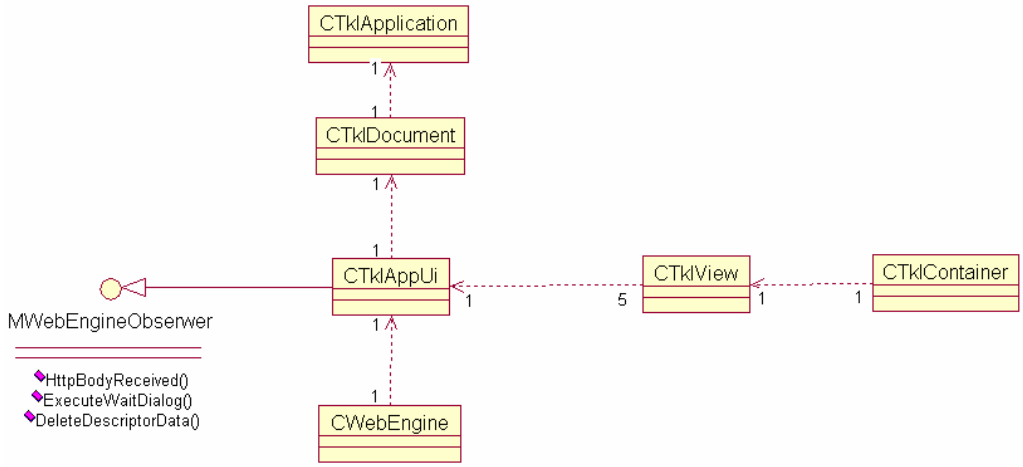
Kuva 19. Käyttötapaukset.

Ohjelma on toteutettu ja testattu S60 versio 3.0:lla käyttäen 356*416 kuvapisteen resoluutiota. Ladatut aikataulutiedot tallennetaan siis saadussa html-muodossa ja ladataan näytölle BrowserControl API:a hyväksi käyttäen. S60 3.0 MR SDK:ssä on ainoastaan vanhemman Palvelut-selaimen moottori, mikä merkitsee, että html-sivut puristetaan näytön levyisiksi ja selaaminen tapahtuu ainoastaan ylös/alas suunnassa.

Mitään suurempaa estettä ohjelman toimimiselle uudemmilla S60 versioilla ei ole, kun selainmoottorin vaihtuessa käytettävä DLL:n nimi pysyy kuitenkin samana. Tällöin sivut näytettäisiin tietokoneen selainta vastaavassa muodossa.

4.2 Arkkitehtuurin valinta ja suunnitteluratkaisut

Kolmesta S60 arkkitehtuurimallista valitsin näkymäarkkitehtuurin (Kuva 20). Tämä oli luonnollinen valinta ohjelmalle, jolla on useita eri näkymiä. Näkymäarkkitehtuurilla siirtyminen näkymien välillä on tehty helpoksi ja käytetty Carbide.c++ teki suurimman osan koodia sitä varten valmiiksi. Erityisesti UI Designerin käyttö nopeutti käyttöliittymän tekoa.



Kuva 20. TKL-ohjelman rakenne.

Ohjelman rakenne noudattaa siis näkymäarkkitehtuurin rakennetta, jonka näimme kuvassa 15. AppUi-luokan kutsumista Engine-luokasta varten on tehty rajapintaluokka, jonka AppUi-luokka perii. Pysäkki- ja linja-

aikataulujen selaamiseen toteutettiin omat näkymät välilehditettynä yhteen. Näin käyttäjän on helppo vaihtaa haluamansa linja- ja pysäkkiaikataulutietojen katsomisen välillä.

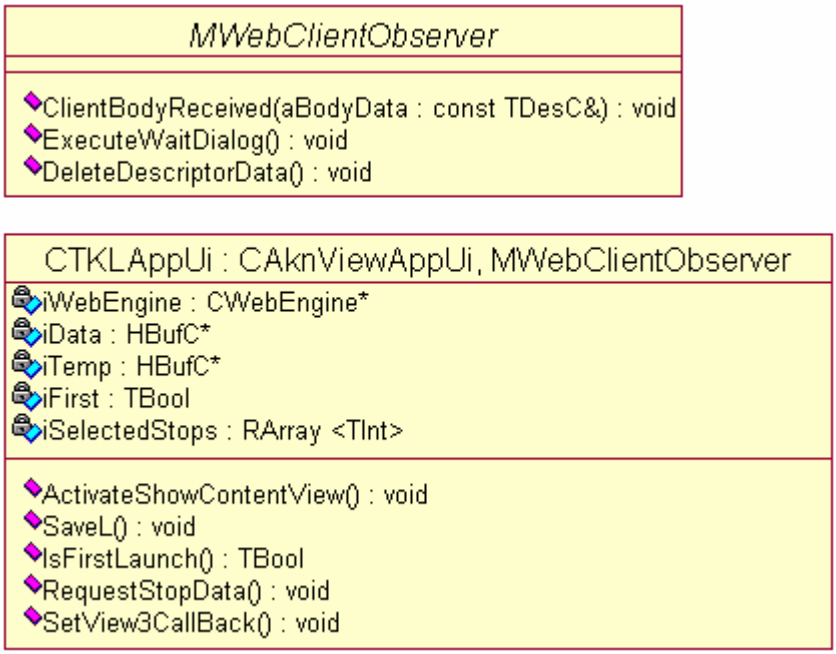
Pysäkkien suuren lukumäärän vuoksi yhtään pysäkkiaikataulutiedostoa ei ladata automaattisesti, vaan käyttäjä voi lisätä haluamansa pysäkit erillisestä näkymästä. Tämän jälkeen ne ovat käytettävissä aina heti. Jokaisen linjan aikataulut ladataan ensimmäisellä käynnistyskerralla niiden vähyyden vuoksi. Niiden lataaminen automaattisesti helpottaa jatkossa myös ohjelman käyttämistä ja pitää sen yksinkertaisempana. Seuraavassa kappaleessa käymme läpi tarkemmin jokaisen luokan toiminnan ja tarkoituksen tarkemmin läpi.

4.3 Luokkien toiminnallisuus

Ensiksi käymme läpi CTklAppUi-luokan toiminnallisuuden. Sen perinteisten tehtävien, kuten valikkokomentojen ja näppäinpainalluskäsittelyn sekä näkymien vaihtelemisen lisäksi sen tehtäväksi jäi muun muassa seuraavaa:

- Tarkistaa onko kaikki linja-aikataulut ladattu koneelle ja käskeä CWebEngine:ä lataamaan ne mikäli ei ole.
- Pyytää CWebEngine:ä lataamaan valittujen pysäkkien aikataulutiedot.
- Ottaa vastaan CWebEngine:n lataaman tiedot ja tallentaa ne.

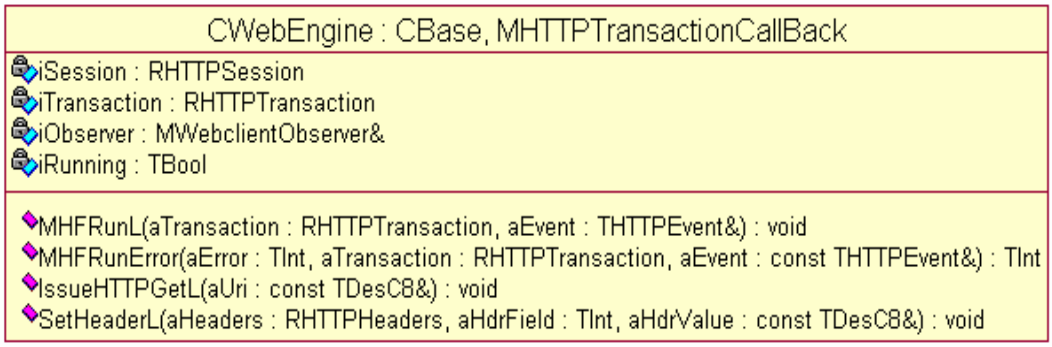
Kuvassa 21 on CTklAppUi-luokan keskeisimmät jäsenmuuttujat sekä funktiot. Lisäksi se toteuttaa kuvassa näkyvän rajapinnan funktiot. Selkeyden vuoksi kuvasta on jätetty pois ohjainluokalle tyypilliset funktiot ja luokkamuuttujat. Ohjainluokan perinteisen toiminnallisuuden voi nähdä muun muassa SDK:n dokumentaatiosta.



Kuva 21. CAppUi:n funktiot ja luokkamuuttujat.

CTKLAppUi-luokkalla on CWebEngine-tyyppinen jäsenmuuttuja, jolle se lähettää pyynnön ladata tietystä URI:sta aikataulutiedot. CWebEngine-luokka lähettää saadut tiedot MWebClientObserv:n kautta CTKLAppUi-luokkalle tallennettavaksi. iTemp-kuvaimeen tallennetaan HTML-dokumentin ”body”-tekstinpalaset, jotka aina lisätään itse iData-muuttujaan, kunhan teksti on sisäisesti oikeassa muodossa. Http-pyyntön valmistuttua tieto tallennetaan iData-kuvaimesta tiedostoon. Näin ladataan jokaisen linjan aikataulutiedot ensimmäisellä käynnistyskerralla.

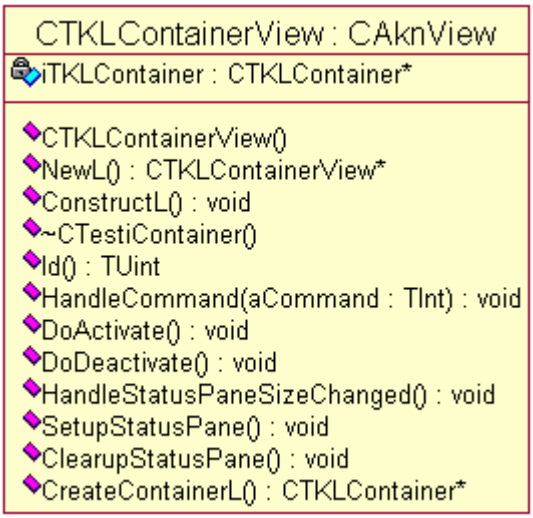
Samaan tapaan CTKLAppUi-luokka pyytää CWebEngine:ä hakemaan lisättyjen pysäkkien tiedot aina kun käyttäjä lisää uusia pysäkkejä ja tallentaa ne tiedostoon. Keskeiset CWebEngine:n funktiot ja luokkamuuttujat näkyvät kuvasta 22.



Kuva 22. CWebEngine:n funktiot ja jäsenmuuttujat.

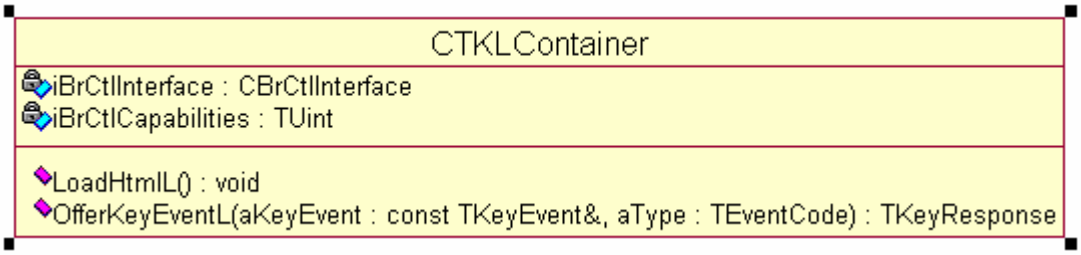
Aliohjainluokkia on 5 kappaletta, joiden elinkaari ulottuu ohjelman käynnistyksestä sen sulkemiseen. Niillä jokaisella on oma näkymäluokka, joka vastaan itse näytölle piirtämisestä ja pitää sisällään käyttöliittymäkontrollit. Nämä näkymäluokat luodaan aina, kun kyseinen näkymä laitetaan päälle ja tuhotaan suljettaessa.

TKL-ohjelmassa näillä aliohjainluokille toteutettu toiminnallisuus on kaikilla lähes samanlainen. Niiden tehtävä on luoda/tuhota näkymäluokka ja lisätä/poistaa ne kontrollipinoon tai pois sieltä. Lisäksi ne ovat kyseisen näkymän valikkokomentojen ensisijainen käsittelypaikka. Kuvassa 23 on ohjelman käynnistys ruudun aliohjain luokan funktiot ja jäsenmuuttujat kuvattuna.



Kuva 23. Näkymäluokan funktiot ja jäsenmuuttuja.

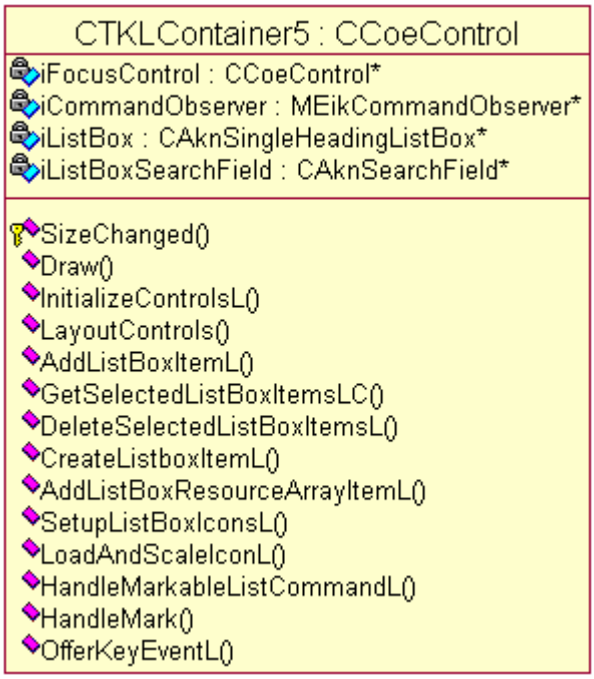
Itse näkymäluokkien toiminnallisuus eroaa toisistaan. CTKLContainer1 & 2 (Kuva 24) on yhdistetty välilehdiksi ja niiden tehtävä on näyttää CTKLContainer3 & 4:sta valittu linjan tai pysäkin aikataulutiedot *Main Pane*:ssa. Tätä varten niillä on CBrCtlInterface-tyyppinen lapsikontrolli, jonka avulla voidaan tallennettu HTML-tiedosto näyttää ruudulla samankaltaisessa muodossa kuin se näkyy internetselaimessa. Lisäksi niillä on S60-ohjelmalle tyypilliset Status- ja Control Pane, jotka rakennetaan resurssitiedostosta.



Kuva 24. CTKLContainer-luokan funktioita ja luokkamuuttujia

LoadHtmlL()-funktion tehtävä on ladata näytölle haluttu tiedosto, kutsumalla *iBrCtlInterface->LoadFileL()*. Näppäinpainallukset ohjataan selainkontrollille *OfferKeyEventL*-funktiossa kutsumalla *iBrCtlInterface->OfferKeyEvent()*. Selainkontrollin kuluttaessa nuolinäppäinten sivuttaispainallukset selaamiseen oli välilehtien vaihtamiseen koodattava toinen näppäin. Paras vaihtoehto käytettävyyden kannalta tähän oli MSK. CTKLConteiner3 & 4 & 5 -luokat sisältävät erilaisia listakontrolleja. Nämä käyttöliittymäkontrollit esiteltiin aikaisemmin luvussa 3.4. CTKLContainer3 & 4 sisältävät erityyliset valintalistakontrollit, kun taas CTKLContainer5 sisältää merkintälistakontrollin. Kaikki sisältävät luonnollisesti ohjelman tyylin mukaiset *Status*- ja *Control Panen* sekä etsintäkentän.

Kuvassa 25 on kuvattu kattavasti CTKLContainer5 luokan funktiot ja jäsenmuuttujat ja siitä saamme hyvän kuvan kaikkien näiden listakontrolleja sisältävien luokkien toiminnallisuudesta. Rakentajia eikä purkajaa ole sisällytetty kuvaan.



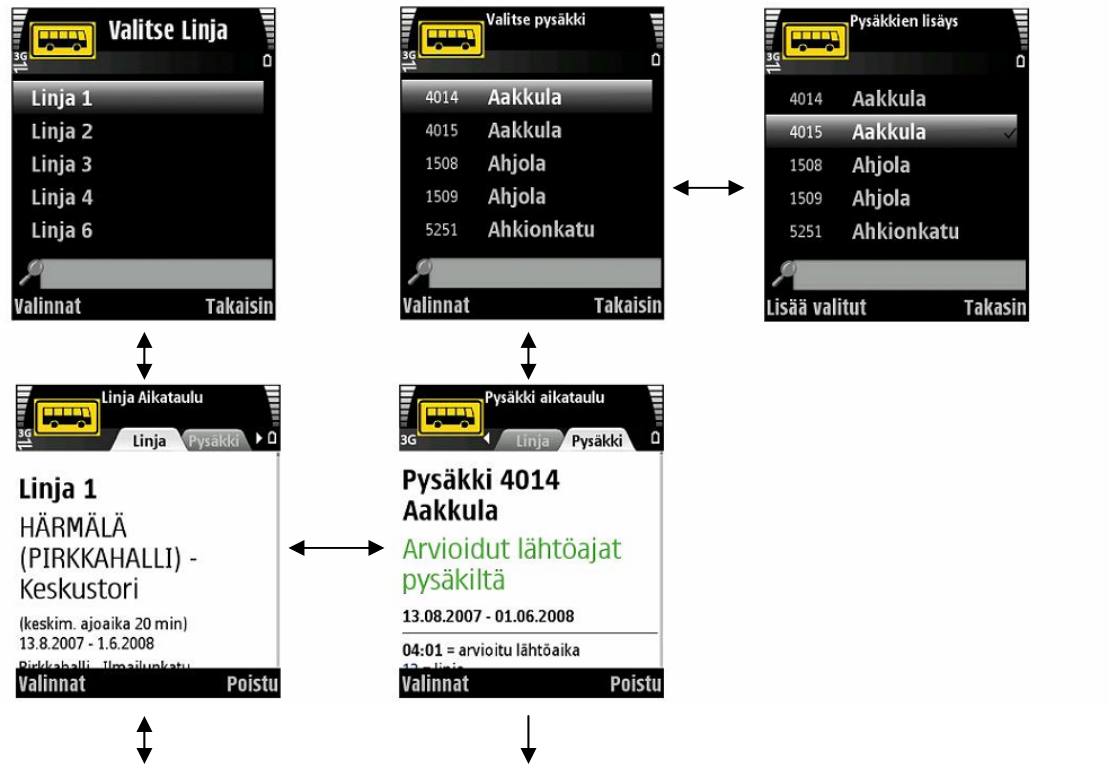
Kuva 25. CTKLContainer5-luokan funktiot ja jäsenmuuttujat.

Funktiot on pyritty nimeämään kuvaaviksi ja suuresta osasta näkee suoraan mitä niillä tehdään. Lyhyesti käytynä ne tekevät:

- *InitializeControls* luo resurssilukijat ja luo resurssitiedostosta listakontrollin ja sen alkiot.
- *AddlistboxResourceArrayItemL* lisää resurssitiedostosta alkion listakontrolliin.
- *CreateListBoxItemL* luo listakontrollin alkion.
- *AddListBoxItemL* lisää alkion listakontrolliin.
- *HandleMarkableListBoxCommandL* on sovelluskehyksen kutsuma funktio, kun käyttäjä tekee merkkauskomennon(Mark, Unmark, Mark all tai UnmarkAll). TKL-ohjelma ei käsittele merkkauksia näillä komennoilla eikä kyseisiä komentoja löydy valikosta.
- *HandleMark* käsittelee merkkauksen, kun käyttäjä painaa MSK:tä.
- *OfferKeyEvent* ohjaa näppäinpainallukset halutuille tahoille, kuten ylös/alas painallukset listakontrollille ja MSK:n painamisen merkkaukset käsittelevälle funktiolle.

4.4 Käyttöliittymä

TKL-ohjelma koostuu viidestä eri ruudusta, joista kaksi välilehditetty. Navigointi ruutujen välillä on pyritty toteuttamaan mahdollisimman loogisesti ja suoraviivaisesti. Kuvassa 26 näkyy ruutujen suhteet toisiinsa ja navigointireitit niiden välillä.



Kuva 26. Navigointi näkymien välillä.

Ohjelman aloitussivuna toimii ensimmäinen välilehtisivu, jonka tehtävä on näyttää linjojen aikatauluja. Ensimmäisellä käynnistyskerralla sen Main Pane jää tyhjäksi. Jatkossa sille tultaessa, ladataan näkyviin viimeisimmän valitun linjan aikataulut tai uuden valitun linjan aikataulut. Kuvassa 27 näkyy ensimmäisen sivun ruutukaappaukset ensimmäisestä käynnistyskerrasta sekä selattaessa linjan 1 aikataulutietoja.

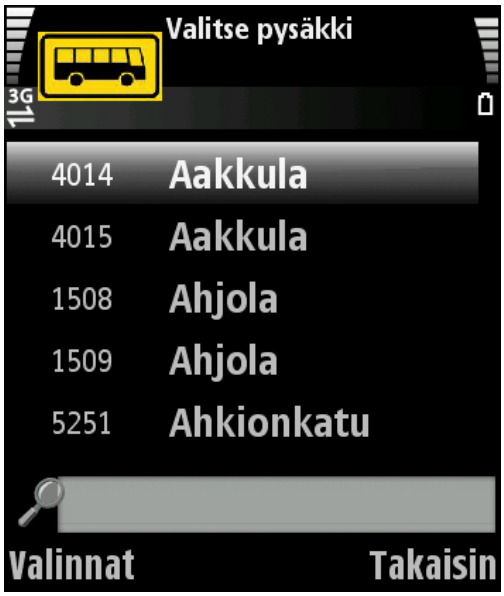
Kuva 27. Ohjelman aloitussivu.

Pysäkkivälilehtisivu on linjavälilehtisivun kaltainen, mutta sen tehtävä on näyttää valitun pysäkin aikataulutiedot. Myös tämä sivu ”muistaa” edellisen valitun pysäkin sille tultaessa, mikäli uutta pysäkkiä ei ole valittu.

Linjanvalintasivu (kuva 28) näyttää kaikki TKL:n linjat, joista käyttäjä voi valita haluamansa näytettäväksi. Valinnan jälkeen ohjelma palaa linjavälilehdelle näyttäen valitun linjan aikataulutiedot.

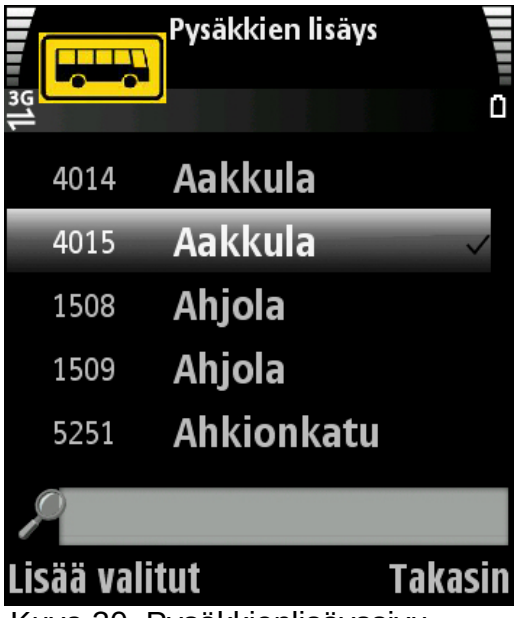
Kuva 28. Linjanvalintasivu.

Pysäkinvalintasivu (Kuva 29) on samanlainen kuin linjanvalintasivu. Aluksi siinä ei tosin ole yhtään pysäkkiä vaan käyttäjän pitää lisätä siihen pysäkkejä pysäkinlisäyssivulta. Kuvassa 25 siihen on ladattu jo pysäkkejä.



Kuva 29. Pysäkinvalintasivu.

Pysäkinlisäyssivulla (Kuva 30) käyttäjä merkitsee haluamansa pysäkin painamalla MSK:ta. Valinnan voi poistaa painamalla sitä uudestaan. Lisäksi sivulla on etsintätoiminto, joten käyttäjän ei tarvitse rullailla pitkää pysäkkilistaa päästäkseen valitsemaan haluamansa pysäkin. Lopuksi käyttäjä painaa LSK:ta ja ohjelma lähtee hakemaan valittuja pysäkkejä ja lisää ne pysäkinlisäyssivulle.



Kuva 30. Pysäkkienlisäyssivu.

4.5 Johtopäätökset

Työ toteuttaa hyvin sen alussa asetetut tavoitteet. Vaikka ohjelma jäi vielä hieman kehittelytasolle, voidaan sen käyttöliittymää ja toiminnallisuutta kuvata onnistuneeksi ja tavoitteiden mukaiseksi.

Käyttöliittymästä tuli selkeä ja helposti navigoitava. Itse aikataulutietojen katsominen samassa muodossa kuin netissä oli myös onnistunut ratkaisu sen toteuttamisen yksinkertaisuuden sekä käytettävyyden kannalta.

UI Designerin käyttö nopeutti käyttöliittymän kehitystä todella paljon. Silla sai helposti ja nopeasti toteutettua tarpeitani vastaavat näkymät ja koodirungon niiden käyttämiseen. Kuitenkin voidaan sanoa, että tämä ei ole totta kaikkien ohjelman kanssa.

Se ei välttämättä aina sovi kuin nopeitten prototyyppi käyttöliittymien toteuttamiseen jokseenkin rajallisen määrän komponenttien takia. Myös sen tekemät koodit ovat alueina muun koodin joukossa, mikä estää muutoksien teon niihin niiden väleihin. Ne ylikirjoitetaan aina uudelleen, mikäli tekee muutoksia UI Designiin. Tämän kiertäminen olisi todella hankalaa ja monesti jopa mahdotonta.

4.6 Jatkokehitysideoita

Ohjelmaa toteuttaessa huomasin jättäneeni paljon engine-puolen toiminnallisuutta, kuten esim. tiedostoon tallentamista AppUi-luokan tehtäväksi. Tällöin kyseinen luokka hieman paisui tekemään periaatteessa sille kuulumattomia tehtäviä, mikä heikensi ohjelman rakenteen selkeyttä. Tämä olisi voitu välttää joko toteuttamalla ko. toiminnallisuus suoraan WebEngine-luokassa tai uudessa luokassa enginen puolella.

Rajoituksia ohjelmalle luo aikataulujen muutokset kahdesti vuodessa. Siihen ei ole tällä hetkellä toteutettu aikataulutietojen päivitystä vaan sen käyttämät web-osoitteet on kovakoodattu. Mahdollisia ratkaisuja tähän

voisi olla uuden käännöksen teko uusituin osoittein tai hieman suurempia koodimuutoksia vaativa, mutta käytännöllisempi oma palvelin johon tarvittavat aikataulutiedot päivitettäisiin ja josta ne voi käydä päivittämässä tarpeen vaatiessa sovellutukseen.

LÄHDELUETTELO

Painetut lähteet

1 Edwards L, Baker R, the Staff of EMMC Software Ltd. Developing Series 60 Applications, A Guide for Symbian OS Developers, 2004.

2 Babin S, Developing Software for Symbian OS, An introduction to Creating Smartphone Applications in C++, 2006.

3 Niskanen Pekka, Symbian ohjelmointi, 2005.

Sähköiset lähteet

3 SDK help for S60 3rd Edition, Maintenance Release.

4 Strings and Descriptors | NewLC. [www-sivu]. [viitattu 27.8.2007] Saatavissa: http://newlc.com/String-and-Descriptors.html

5 Introduction to RBuf. [pdf-tiedosto]. [viitattu 18.9.2007] Saatavissa:http://developer.symbian.com/main/downloads/papers/RBuf/Introduction_to_RBuf_v1.0.pdf

6 LifeCycleModel. [gif-kuva]. [viitattu 18.9.2007] Saatavissa: http://www.symbian.com/developer/techlib/v9.1docs/doc_source/guide/base-subsystemguide/n10086/InterProcessCommunication/AsynchronousServicesGuide/AsynchronousServicesGuide3/LifeCycleModel.gif

7 Device Specifications | Forum Nokia. [www-sivu]. [viitattu 1.10.2007] Saatavissa: http://www.forum.nokia.com/devices/matrix_all_1.html

8 Capabilities Granting. [www-sivu]. [viitattu 5.10.2007] Saatavissa: http://www.forum.nokia.com/main/technical_ services/ testing/cap_granting.html

9 S60 UI Style Guide v1.3. [pdf-tiedosto]. [viitattu 7.10.2007] Saatavissa: http://sw.nokia.com/id/04c58d5a-84c3-42db-83d5-486c1cf3e6b3/S60_UI_Style_Guide_v1_3_en.pdf

10 S60 application framework handbook. [pdf-tiedosto]. [viitattu 9.10.2007] Saatavissa: http://sw.nokia.com/id/8cc94f4a-cbd9-4f82-b4d1-790b7569733e/S60_Platform_Application_Framework_Handbook_v2_0_en.pdf

11 Symbian OS View Architecture v1.1. [pdf-tiedosto]. [viitattu 09.12.2007] Saatavissa: http://sw.nokia.com/id/3cab67ad-db01-400a-9467-91b8be7ccbba/Symbian_OS_View_Architecture_v1_1_en.pdf

12 S60 Platform Avkon UI Resources Listboxes v1.1. [pdf-tiedosto]. [viitattu 12.01.2008] Saatavissa: http://sw.nokia.com/id/0ef53656-8e0d-48f2-8b67-1ebed12d526d/S60_Platform_Avkon_UI_Resources_v1_1_en.zip

13 Symbian OS platform capability descriptions. [pdf-tiedosto]. [viitattu 06.02.2008] Saatavissa: http://sw.nokia.com/id/6bde04b0-041f-4bc0-8104-40ebf956aa1d/Capability_Descriptions_v1_0_en.pdf

14 Overview to Symbian OS security. [pdf-tiedosto]. [viitattu 06.02.2008] Saatavissa: http://sw.nokia.com/id/5e713b29-fe0e-488d-8fc6-b4dd1950f3c2/Symbian_OS_Overview_To_Security_v1_1_en.pdf

15 The Complete Guide to Signing. [pdf-tiedosto]. [viitattu 10.02.2008] Saatavissa: http://developer.symbian.com/main/downloads/files/TheCompleteGuideToSymbianSigned.zip